KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Janne Hakkarainen

TYÖKALUJÄRJESTELMÄN PROTOTYYPIN TOTEUTTAMINEN KÄYTTÄEN JAVA-OHJELMOINTIKIELTÄ

Opinnäytetyö
Toukokuu 2013

OPINNÄYTETYÖ
Toukokuu 2013
Tietojenkäsittelyn koulutusohjelma

Karjalankatu 3
80200 JOENSUU
p. 013 260 6800

Tekijä
Janne Hakkarainen

Nimeke
Työkalujärjestelmän prototyypin toteuttaminen käyttäen Java-ohjelmointikieltä

Toimeksiantaja
Karelia-ammattikorkeakoulu, tietojenkäsittelyn koulutusohjelma

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa prototyyppiasteinen web-sovelluksena toimiva tietovaranto Karelia-ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmalle. Kehitetyn sovelluksen tarkoituksena on tukea eri vuosina tehtäviä ohjelmistoprojekteja, kurssien harjoitustöitä ja muuta ohjelmoinnin oppimista. Työkalujärjestelmään tuli olla mahdollista tallentaa tietoa erilaisista ohjelmistokehitykseen käytettävistä työkaluista ja myös hakea tietoa niistä.

Opinnäytetyön tarkoituksena oli myös tarkastella web-pohjaisen tietojärjestelmän kehittämistä pelkästään Java-ohjelmointikielellä. Lisäksi pyrkimyksenä oli tehdä järjestelmästä mahdollisimman oliopohjainen ja riippumaton käytettävästä tietokannanhallintajärjestelmästä. Prototyypin kehittämiseen käytettiin Vaadin-, Hibernate- ja SimpleCaptcha-sovelluskehyksiä. Tietokannanhallintajärjestelmänä käytettiin MySQL:ää.

Lopputuloksena syntyi tietojärjestelmän prototyyppi, josta toimeksiantaja voi halutessaan kehittää valmiin tietojärjestelmän tietojenkäsittelyn koulutusohjelman käyttöön. Vaadin-sovelluskehys mahdollisti järjestelmän käyttöliittymän toteuttamisen käyttäen pelkkää Java-ohjelmointikieltä. Hibernate mahdollisti olioiden tietojen tallentamisen tietokantaan ilman, että kehittämisessä tarvitsi käyttää kyselykieliä.

Kieli
suomi

Sivuja 51

Asiasanat
web-sovellus, tietokantasovellus, ohjelmistokehitys

THESIS
May 2013
Degree Programme in Business Information Technology
Karjalankatu 3
FI 80200 JOENSUU FINLAND
Tel. +35813 260 6800

Author

Janne Hakkarainen

Title
Implementation of the Tool Information System using Java Programming Language

Commissioned by
Karelia University of Applied Sciences in Business Information Technology

Abstract

The aim of this thesis was to implement a prototype of a web-based information system for the Degree Programme in Business Information Technology of Karelia University of Applied Sciences. The purpose of the implemented web-application is to support software projects, course assignments and to help students to learn how to program. The system was required to allow saving and retrieving data about different software development tools.

A further purpose of this thesis was to examine the possibility of developing a web-based information system using only Java programming language for development. In addition, the aim was to make the system design to follow object-oriented pattern and make it independent of underlying database management system. Vaadin, Hibernate and SimpleCaptcha Frameworks were used for the development. MySQL was used as a database management system.

The end result was a prototype of an information system, from which the client may choose to develop a complete system for the Degree Programme in Business Information Technology. Vaadin Framework made possible to develop the user interface of the system using only the Java programming language. Hibernate enabled storing the object data to the database without using query languages.

Language
Finnish

Pages 51

Keywords

web application, database application, software development

Sisältö

Lyhenteet ja käsitteet

Android	Suosittu käyttöjärjestelmä, jonka eri versioita on useissa eri mobiililaitteissa.
CRUD	Create, Read, Update, Delete tarkoittaa tiedon lisäämistä, lukemista, päivittämistä ja poistamista tietokannasta.
CSS	Cascading Style Sheets, kieli ja tyyliohje, jolla kuvataan, miten nettisivujen tulisi näkyä selaimessa.
Injektio	Injektio yleisellä tasolla tarkoittaa mm. erilaisten hyödyllisten tai haitallisten kielten ujuttamista koodin sekaan sellaisessa paikassa sovellusta, missä se on mahdollista esimerkiksi tietoturva-aukon kautta.
ORM	Object Relational Mapping, tekniikka olioiden ominaisuuksien ja tietokannan kenttien yhdistämiseen ja tietokantaoperaatioiden automatisointiin.
Palvelinpuoli	Palvelinpuoli tietojärjestelmästä, jossa useiden web-sovellusten tapauksissa prosessoidaan Internet-selaimen kautta lähetetty tieto. Palvelinpuolella toimii myös tietokannan hallintajärjestelmä.
Persistointi	Tarkoittaa ohjelman ajon aikaisen olion tilan tallentamista tietokantaan, josta se on myöhemmin ladattavissa takaisin ohjelmaan.
Request-Response	Pyyntö ja vastaus, yksi yleinen tapa tietojärjestelmän selainpuolen ja palvelinpuolen keskinäiseen kommunikointiin.
Selainpuoli	Viittaa web-sovelluksen Internet-selaimeen tai muuhun asiakassovellukseen.

Sovelluskehys	Joukko kirjastoja, jotka sisältävät ohjelmistokehityksessä hyödyllisiä luokkia, joiden tarkoitus on helpottaa ohjelmistotuotantoa siten, ettei ohjelmistokehittäjien tarvitse suunnitella, määritellä, toteuttaa ja testata koodia niin paljon.
SQL	Structured Query Language, lähes kaikkien tietokannan hallintajärjestelmien ymmärtämä standardoitu kyselykieli, jolla voi tehdä erilaisia tietokantaoperaatioita.
SQL-murre	Poikkeaa standardin mukaisesta SQL-kielestä. Esimerkiksi MySQL tietokannan hallintajärjestelmässä on oma SQL-murre, joka mahdollistaa mm. joidenkin tietokantaoperaatioiden tekemisen eri tavalla kuin standardin mukaisessa SQL-kielessä.
Työkaluvihje	Käyttöliittymäkomponentin käyttöä helpottava opasteteksti.
Validointi	Sovelluksen käyttöliittymään syötettyjen tietojen kelvollisuuden tarkistus, joka voidaan tehdä joko selain- tai palvelinpäässä.
Web-lomakeet	Web-sovelluksissa käytetty käyttöliittymäratkaisu käyttäjän syötteiden lukemiseksi ja palvelimelle prosessoitavaksi välittämiseksi.

1 Johdanto

Opinnäytetyön toimeksiantona oli tehdä Internet-selaimessa toimivan wiki-tyyppisen tietojärjestelmän prototyyppi Karelia-ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmalle. Järjestelmän tarkoitus on olla käyttäjille avoin ja helppokäyttöinen. Tietojärjestelmän ideana on auttaa opiskelijoita valitsemaan sopivia työkaluja erilaisiin projekteihin. Toimeksiantona tehtävää tietojärjestelmän prototyyppiä kutsutaan tässä opinnäytetyössä työkalujärjestelmäksi ja sinne taltioitavia ohjelmistokehitysvälineitä, sovelluskehyksiä ym. kutsutaan työkaluiksi.

Ennen työkalujärjestelmän kehityksen aloittamista ohjelmointikieleksi valikoitui Java. Halusin pitää järjestelmän kehittämisen mahdollisimman korkealla tasolla ja saada enemmän kokemusta sovelluskehyksistä, joten järjestelmän toteuttamiseen käytin Vaadin- ja Hibernate-sovelluskehyksiä. Alun perin olin ajatellut käyttää myös Spring-sovelluskehystä, mutta päädyin jättämään sen pois mm. aikataulullisista syistä.

Työkalujärjestelmän kehityksen edetessä kiinnostuin siitä, miten pitkälle koko tietojärjestelmän voisi toteuttaa käyttäen pelkkää Java-ohjelmointikieltä. Halusin myös tietää, että voisiko sen jopa kokonaan toteuttaa ilman, että järjestelmässä tarvitsee käyttää web-sovelluksille tyypillisiä merkkaus- ja kyselykieliä tai muita tekniikoita ohjelmoijan toimesta.

Tietokannanhallintajärjestelmäksi valikoitui MySQL, koska se sopii hyvin projektin avoimeen linjaukseen. Koska työkalujärjestelmässä käytetyn Javan filosofiaan kuuluu järjestelmäriippumattomuus, pyrin tietynlaisen harmonian vuoksi tekemään työkalujärjestelmästä riippumattoman myös tietokannanhallintajärjestelmästä soveltamalla Hibernatea.

2 Tietoperusta ja teoria

2.1 Työkalujärjestelmältä vaaditut ominaisuudet

Toteutettavan järjestelmän prototyypin tulee olla tietokantapohjainen ja web-sovelluksena toimiva usean käyttäjän portaali. Sen keskeisinä vaatimuksina on olla helposti selattavissa ja muutenkin käytettävyydeltään kohtuullisen helppo erityisesti tiedon selaamisen osalta. Prototyypissä tulee olla myös web-lomakkeiden validointi, josta mahdollisen jatkokehittäjän on helppo jatkaa järjestelmän kehittämistä asettamalla esimerkiksi tiukempia läpäisyvaatimuksia web-lomakkeella lähetetylle tiedolle.

Työkalujärjestelmän toiminnallisuuden toteuttaminen oli melko vapaata ja toteutettavien ominaisuuksien toteuttamistavassa ei ollut erityisiä vaatimuksia. Koska järjestelmän on tarkoitus olla hyvä käytettävyydeltään, on järkevää noudattaa hyväksi todettuja käytäntöjä käyttöliittymän toteuttamisessa.

Työkalujärjestelmän tarkoituksena on tukea mm. eri vuosina tehtäviä ohjelmistoprojekteja, kurssien harjoitustöitä ja ohjelmoinnin oppimista. Varantoon on tarkoitus kasata tietoa erilaisista työkaluista liittyen eri ohjelmointikieliin, kuten Java, PHP, C# yms. Työkalujärjestelmään tallennettujen työkalujen tiedon tulisi olla helposti selattavissa, lisättävissä ja päivitettävissä. Järjestelmän käyttöliittymän tulee toimia Internet-selaimessa ja muun prosessoinnin tulee tapahtua palvelimella.

Työkalujärjestelmään tallennetun tiedon olisi tarkoitus olla hyödynnettävissä mm. kursseilla tehtävien projektien alussa. Kun esimerkiksi projektiryhmä saa toimeksiannon ja tekee sen purkamisen kautta päätöksen käytettävistä tekniikoista, voisi työkalujen tietoja sisältävä tietovaranto olla tukena valitsemassa sopivia työkaluja. Työkaluja voisi hakea esimerkiksi valintaruutuja klikkailemalla.

2.2 All-Java-paradigma ja siirrettävyys

Tässä opinnäytetyössä termi all-Java viittaa siihen, että ohjelmisto on toteutettu kokonaan Java-ohjelmointikieltä käyttäen, eikä sen tekemiseen ole käytetty lisäksi esimerkiksi JavaScript-ohjelmointikieltä, HTML-merkkauskieltä tai SQL-kyselykieltä. Toimintamallina on ollut myös noudattaa hyvää olio-ohjelmointi-lähestymistapaa.

All-Java-paradigman takia oli jokseenkin todennäköistä, että työkalujärjestelmässä jouduttaisiin luopumaan joistakin ominaisuuksista ja esteettisistä elementeistä, jotta prototyypin toteuttaminen onnistuisi. Alkuvaiheessa ei myöskään ollut varmuutta siitä, onko mahdollista välttyä käyttämästä muita kieliä järjestelmän toteuttamisessa. Lisäksi oli odotettavissa, että jotkin asiat olisi mahdollisesti helpompaa tehdä jollakin muulla tekniikalla kuin Javalla (esim. HTML ja SQL).

Järjestelmän käyttöliittymän tekeminen Java-ohjelmointikielellä ilman sovelluskehyksiä olisi ollut käytännössä mahdotonta, koska Internet-selaimet eivät pysty näyttämään pelkällä Javalla koodattuja sivuja. Tämän vuoksi on tarpeen käyttää jotakin sovelluskehystä, joka muuttaa Java-koodin Internet-selaimen ymmärtämään muotoon. Yksi esimerkki tällaisesta sovelluskehyksestä on Vaadin, jota projektissa on käytetty. Näin ollen todellisuudessa eri kieliä ja tekniikoita käytetään Vaatimen pinnan alla, mutta ohjelmoijan ei tarvitse niitä suoraan käyttää tai niitä ymmärtää.

Järjestelmässä tarvittavien tietokantaoperaatioiden tekeminen Javalla sisällyttämättä koodiin kyselykieltä ei ole mahdollista ilman työkalua, joka tekee sen ohjelmoijan puolesta. Tämän vuoksi työkalujärjestelmässä on käytetty Hibernate-sovelluskehystä tietokantaoperaatioiden tekemiseen. Hibernate vaikutti mahdollistavan olioiden tietojen persistoinnin tietokantaan Java-ohjelmointikieltä käyttäen.

Javan luonteeseen on kuulunut siirrettävyys ja alustariippumattomuus, joten all-Java-paradigman jälkeen toissijaisena pyrkimyksenä työkalujärjestelmässä on pyritty myös riippumattomuuteen tietokannan hallintajärjestelmästä. Sen saavuttamiseen Hibernate vaikutti tarjoavan mahdollisuuden.

2.3 Java-ohjelmistoalusta

2.3.1 Esittely

Java on Sun Microsystemsin kehittämä vuonna 1995 julkaistu ohjelmistoalusta, joka koostuu useasta eri osasta. Java-teknologiaan kuuluu sekä Java-ohjelmistoalusta että Java-ohjelmointikieli, jolla voi kehittää ohjelmistoja Java-alustalle. Tähän päivään mennessä tehdyistä ohjelmistoalustoista on kirjoitettu tarkemmin kohdassa 2.3.8.

Yksi Java-alustan päätarkoituksista on suorittaa Java-ohjelmia missä tahansa käyttöjärjestelmissä, joihin on saatavilla ja asennettu Java Runtime Environment. Tämän takia Java-ohjelmia ei pitäisi joutua muokkaamaan eikä kääntämään uudelleen, jotta ne toimisivat eri järjestelmissä. (Oracle 2013a.)

2.3.2 Java-virtuaalikone

Java-virtuaalikone (JVM) on Java-alustan kulmakivi. Java-virtuaalikone mahdollistaa Java-sovellusten suorittamisen. Java-virtuaalikone on tyypillisesti asennettu jollekin käyttöjärjestelmälle, mutta se on myös mahdollista toteuttaa suoraan tietokonelaitteistoon. Java-virtuaalikone suorittaa class-tiedostoissa olevaa tavukoodia, jota saa kääntämällä ohjelmoijan kirjoittamaa koodia jollakin kääntäjällä, ohjelmointikielestä riippuen. (Lindholm, Yellin, Bracha & Buckley 2013, 2.)

Java-virtuaalikone tuo mukanaan muitakin hyötyjä kuin riippumattomuuden käyttöjärjestelmästä. Yksi suuri hyöty on se, että virtuaalikone mahdollistaa turvallisemman muistin hallinnan, koska ohjelmoijan ei tarvitse vapauttaa dynaamisesti varattua muistia itse. Automaattinen muistinhallinta, jota kutsutaan myös roskien keruuksi, pitää huolen olioiden muistin vapauttamisesta. (Lindholm ym. 2013, 13.)

2.3.3 JRE ja JDK

Saadakseen Java-sovelluksen toimimaan tulee tietokoneen käyttäjän ladata koneelleen ilmainen JRE eli Java Runtime Environment. JRE voi tulla myös Java-sovelluksen mukana. JRE sisältää kaiken tarvittavan, kuten Java-virtuaalikoneen, suorittamaan Java-sovelluksia. Virtuaalikoneen lisäksi JRE:n mukana tulee mm. Java Web Start Launcher -sovellus. (Oracle 2012a.)

JDK eli Java Development Kit on ohjelmistokehittäjille tarkoitettu kokonaisuus, joka sisältää mm. Java-kääntäjän ja JRE:n. JDK:n mukana tulee myös joitakin muita hyödyllisiä työkaluja Java-sovellusten ja sovelmien tekemiseen ja testaukseen. Näitä ovat mm. Javadoc ja Jar, jotka esitellään tämän luvun myöhemmissä osissa. (Oracle 2012b.)

2.3.4 Java-ohjelmointikieli

Osana Java-kokonaisuuteen kuuluu Java-ohjelmointikieli, joka on yleiskäyttöinen korkean tason ohjelmointikieli (Gosling, Joy, Steele, Bracha & Buckley 2013, 1). Useimmat Java-tavukoodiksi käännetyt ohjelmat on kirjoitettu Java-ohjelmointikielellä. JDK:n mukana tuleva Java-kääntäjä kääntää ainoastaan Javalla kirjoitettua koodia, joten Java on varsin looginen valinta ohjelmoida Java-sovelluksia.

Java-ohjelmointikieli muistuttaa syntaksiltaan paljon C- ja C++-kieliä, joiden lisäksi se on saanut vaikutteita muistakin ohjelmointikielistä. Java on suunnattu erityisesti tuotantoon, joten siinä on vältetty tutkimustarkoituksiin tehdyille kielille tavallisia uusia ja huonosti testattuja ominaisuuksia. (Gosling ym. 2013, 1.)

2.3.5 Luokat ja oliot

Java-ohjelmointikieli on luokkiin perustuva (class-based) olio-ohjelmointikieli (object-oriented programming language). Java-ohjelmoinnissa olio-ohjelmointi-paradigmaa noudattaen saadaan koodista selkeää ja uudelleen käytettävää. (Gosling ym. 2013, 1.) Oliomaista ohjelmointitapaa tarvitaan työkalujärjestelmässä koodin ylläpidollisten syi-

den lisäksi myös toteuttamisessa käytettävien sovelluskehysten Vaatimen ja Hibernaten takia.

Java-luokka määritellään .java-päätteisiin tiedostoihin ja tiedostoissa on usein erilaisia metodeja ja muodostin, jota kutsutaan silloin, kun luokasta muodostetaan olio. Java-luokasta ei ole välttämätöntä luoda oliota vaan luokkiin kirjoitettuja metodeja ja muuttujia voi käyttää, mikäli ne määritellään staattisiksi (static). Koodissa olio (object) luodaan luokasta käyttämällä new-operaattoria, jolloin sovellukselle varataan muistia olion tarvitseman määrän verran. (Gosling ym. 2013, 205–206.)

Java-ohjelmoinnissa luokkia on mahdollista periyttää muista luokista. Tällöin perivä luokka perii mm. kaikki perittävän luokan julkiset (public) ominaisuudet. Perittyjen ominaisuuksien lisäksi perivään luokkaan on mahdollista lisätä uusia ominaisuuksia ja ylikirjoittaa perittävän luokan ominaisuuksia. Työkalujärjestelmän kehityksessä periyttämistä on käytetty paljon mm. uusia komponentteja kehitettäessä ja niiden pohjana on käytetty Vaatimen komponentteja, jotka ovat samalla myös luokkia. Esimerkiksi Vaatimen VerticalLayout-luokasta on periytetty ToolInfoContainer-luokka (kuva 1), johon on VerticalLayoutin ominaisuuksien lisäksi lisätty muuta toiminnallisuutta.

```
package fi.karelia.ont.jh.components;

import java.util.List;

public class ToolInfoContainer extends VerticalLayout {

    public ToolInfoContainer(List records, String header, Form form, Navigator navigator,Database db) {

        if (header != null) {
            this.addComponent(new Label(header));
        }
        if (records == null) {
            System.err.println("Bugi, lista on tyhjä.");
        } else {

            for (int i = 0; i < records.size(); i++) {
                Tool tool = (Tool) records.get(i);
                ToolInfo toolInfo = new ToolInfo(tool,form,navigator,db);
                this.addComponent(toolInfo);
            }

        }
    }
}
```

Kuva 1. Esimerkki työkalujärjestelmään tehdystä pienestä ToolInfoContainer-luokasta, joka perii ominaisuudet VerticalLayout-luokalta extends-avainsanalla luokan määrittelyssä.

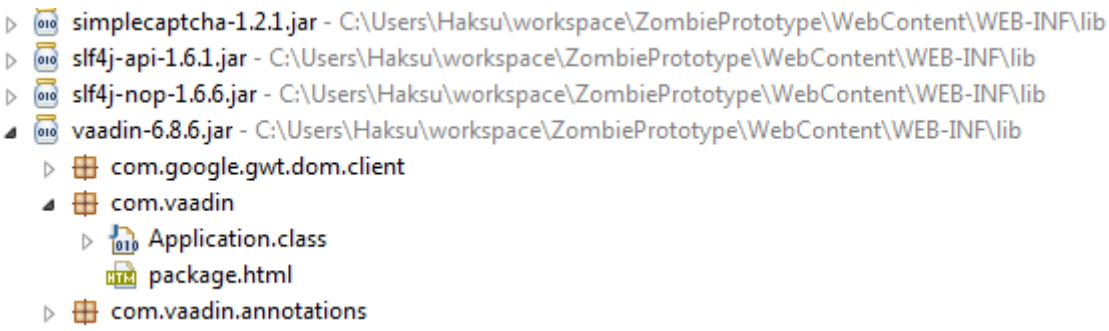
Tässä opinnäytetyössä luokasta ja oliosta puhuttaessa tarkoitetaan samaa, koska työkalujärjestelmän toteuttamisessa on vältetty staattisten muuttujien ja metodien käyttöä.

Lähes kaikista luokista on järjestelmässä luotu olio. Myös Vaatimen käyttöliittymä- ja asettelukomponentteja käytetään kuin olioita, joten niitäkin kutsutaan tässä usein olioiksi.

2.3.6 Kirjastot, paketit ja jar-paketit

Javassa kirjasto (library) koostuu Java-luokista. Tyypillisesti Java-luokat ovat yhdessä tai useammassa paketissa (kuva 2), joita käytetään mm. Java-luokkien ryhmittelyyn. Pakettien avulla on myös mm. mahdollista rajoittaa koodin käyttöä muualta koodista. (Gosling ym. 2013, 154–155.) Javan standardikirjastot, johon sisältyy ohjelmointia helpottavia luokkia, tulevat JRE:n mukana (Oracle 2012b).

Usein kirjastot ja kokonaiset sovelluskehykset pakataan yleisesti tunnettujen zip-tiedostojen tapaan jar-tiedostoihin (kuva 2), mikä mahdollistaa mm. kaikkien tiedostojen tiivistämisen yhteen tiedostoon. Sen lisäksi, että se helpottaa ohjelmointia, myös ohjelman suorituskyky saattaa parantua joissakin tilanteissa. Jar-tiedostoihin voi pakata muitakin resursseja, kuten esimerkiksi kuva-tiedostoja. (Oracle 2012c.)



Kuva 2. Application-luokka com.vaadin-paketissa, joka sijaitsee vaadin-6.8.6-nimisessä jar-paketissa.

2.3.7 Kääntäjä ja Java-tavukoodi

Java-kääntäjän tehtävä on lukea lähdekooditiedostot ja kääntää ne Java-virtuaalikoneen ymmärtämäksi tavukoodiksi. Java Development Kitin mukana tulee Java-kääntäjä Javac (Oracle 2013b). Java-sovelluksia voi kirjoittaa myös muilla ohjelmointikielillä, jos so-

piva kääntäjä löytyy. Esimerkiksi Pascal-ohjelmointikielellä kirjoitettua koodia on mahdollista kääntää Free Pascal -kääntäjällä Java-tavukoodiksi. (Free Pascal 2012.)

2.3.8 Javan eri editiot

Erilaisia virallisia Java-ohjelmistoalustoja on tähän päivään mennessä ilmestynyt neljä, joita ovat Java SE, Java EE, Java ME ja JavaFX. Eri Java-ohjelmistoalustat on suunnattu erilaisiin tarkoituksiin. (Oracle 2012d.)

Java SE eli Java Standard Edition sisältää Java-ohjelmointikielen ydintoiminnallisuudet, kuten Java SE API:n, virtuaalikoneen ja kehitystyökalut, joita tarvitaan Java-teknologiaa hyödyntävissä ohjelmistoissa. Java SE API:sta löytyy kaikki Javan perustyypeistä ja olioista korkean tason luokkiin, joita käytetään verkkoyhteystoimintoihin, tietoturvaan, tietokantaoperaatioihin, grafiikkaan ja XML:n jäsentämiseen. (Oracle 2012d.)

Java ME eli Java Micro Edition on Sun Microsystemsin kehittämä Java-alusta järjestelmille, kuten kännyköille. Java ME -alustaan kuuluu virtuaalikone ja API, jotka on optimoitu resursseiltaan vähäisiä laitteita varten. (Oracle 2012d.) Java ME -alustakin jakautuu erilaisiin kokoonpanoihin, kuten Connected Limited Device Configuration (CLDC) ja Connected Device Configuration (CDC). CLDC:n kirjastot ja virtuaalikone on tarkoitettu resursseiltaan rajoittuneemmille mobiililaitteille, kun taas CDC on suunnattu tehokkaammille mobiililaitteille, kuten älypuhelimille. (Oracle 2013c.)

JavaFX on ohjelmistoalusta RIA-sovellusten (Rich Internet Application) luomiseen. JavaFX ottaa hyvin hyödyn irti selainta käyttävän koneen laitteistosta mahdollistaen mm. modernin ja prosessointitehoa vaativan grafiikan käytön. JavaFX olisi saattanut olla toimiva vaihtoehto Vaatimelle työkalujärjestelmän kehityksessä, mutta sen soveltuminen all-Java-paradigmaan ei ollut varmaa. (Oracle 2012d.)

Java EE eli Java Enterprise Edition on Java SE:n päälle rakennettu alusta. Java EE:n mukana tulee palvelinpuolen ohjelmointia helpottavia kirjastoja. Yksi merkittävä web-komponentti (ns. web component) Java EE-ympäristössä on Java-luokka Servlet, joka

dynaamisesti prosessoi pyynnöt (requests) ja muodostaa vastaukset (responses). (Oracle 2012d; Oracle 2013d.) Koska toimeksiantona toteutetun työkalujärjestelmän tekemiseen on käytetty Vaadin-sovelluskehystä ja se toimii Java-web-palvelimella, on siinä käytetty Java EE -ohjelmistokehitysalustaa ja Servlet-teknologiaa Java-ohjelmistokehitykseen.

2.3.9 AWT ja Swing

AWT eli Abstract Window Toolkit on Javan alkuperäinen graafisten sovellusten kehittämiseen käytettävä Java-kirjasto. AWT mahdollistaa erilaisten graafisten käyttöliittymien tekemisen esimerkiksi työpöytäsovelluksiin. AWT-komponentit kuten valintaruudut näkyvät esimerkiksi Windows-käyttöjärjestelmässä eri tavalla kuin muissa käyttöjärjestelmissä, koska AWT-sovellukset käyttävät käyttöjärjestelmän omia komponentteja. (Oracle 2011a.)

AWT:n jälkeen ilmestyneet Swing-komponentit korjasivat joitakin AWT:n aiheuttamia ongelmia, kuten käyttöliittymäkomponenttien ulkonäölliset erot eri käyttöjärjestelmien välillä. Swing-komponentit ovat kokonaan Javalla kehitettyjä käyttöliittymäkomponentteja ja näin ollen niiden ulkonäkö ei ole automaattisesti riippuvainen käyttöjärjestelmästä. (Oracle 2011b.)

2.3.10 Tietorakenteet

Java-ohjelmointikielen standardikirjastoihin kuuluu joukko tietorakenteita, joista työkalujärjestelmän kehityksessä käytetään mm. taulukkolistoja (ArrayLists). Tietorakenteisiin kuten taulukkolistoihin voi varastoida eri olioita, mikä helpottaa olioiden käsittelemistä ja kuljettamista koodissa esimerkiksi jollekin metodille välittämällä ne parametreina. (Oracle 2011c.)

2.3.11 Annotaatiot

Java SE:n versiosta 5.0 lähtien Java-ohjelmointikieleen tuli mahdolliseksi annotaatioiden (annotations) lisääminen koodiin (Gosling ym. 2013, xvii). Annotaatiot ovat tietyntyyppistä metadataa, ja ne mahdollistavat esimerkiksi Hibernatea käyttäessä tietokantataulujen luomisen suoraan Java-luokista, mitä on käytetty työkalujärjestelmänkin tapauksessa. Annotaatioita voi käyttää moniin erilaisiin tarkoituksiin.

Java-luokissa olevat annotaatiot tunnistaa siitä, että ne alkavat @-merkillä. Jos halutaan esimerkiksi luoda tietokantataulu Java-luokista Hibernatella, täytyy koodissa olevan luokan määrittelyn yläpuolelle kirjoittaa annotaatio @Entity. Historiatietojen käyttöön ottamiseksi täytyy kirjoittaa samaiseen paikkaan @Audited. Annotaatio @Id puolestaan kertoo sen, mikä kenttä tietokannassa on id. Kuvassa 3 on havainnollistettu annotaatioiden käyttämistä työkalujärjestelmän Tag-luokassa.

```
package fi.karelia.ont.jh.tables;

import java.io.Serializable;

@Entity
@Audited
public class Tag implements Serializable {

    private int idTag;
    private String name;
    private String type;
    private Timestamp time;

    public Tag() {}

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int getIdTag() {
        return idTag;
    }
```

Kuva 3. Esimerkkejä annotaatioista Tag-luokassa.

2.3.12 Javadoc

Yleinen tapa Java-sovellusten dokumentointiin on kirjoittaa dokumentaatio suoraan lähdekoodiin Java-kommenttimerkkien sisään. Javadoc-työkalu generoi HTML-muodossa olevan API-dokumentaation Java-koodiin kirjoitetuista kommenteista. (Ora-

cle 2013e.) HTML-dokumentaation lisäksi javadoc-kommentit näkyvät myös kätevästi esim. Eclipse-kehitysympäristössä, jota on käytetty työkalujärjestelmän kehittämiseen. Työkalujärjestelmässä javadoc-kommentteja on pyritty lisäämään kaikkiin luokkien määrittelyn, muodostimen ja etenkin kaikkein monimutkaisimpien metodien yläpuolelle.

2.3.13 Javan kaltaisia toteutuksia

Javan mielekkyydestä ohjelmointikielenä kertonee se, että Android-sovelluksien ohjelmointiin käytetään Java-ohjelmointikieltä. Android myös käyttää Javan tavoin virtuaalikonetta. Androidin Dalvik-virtuaalikone on optimoitu akulla toimiville laitteille, joissa on pöytäkoneisiin nähden rajoitetusti muistia ja prosessoritehoa. (Google 2008a; Google 2008b.) Käyttöliittymän tekeminen Android-sovellukseen on melko helppoa Java-ohjelmoijalta, jos on ennestään käyttänyt esimerkiksi AWT:tä tai Swingiä.

Google Web Toolkit (GWT) on joukko työkaluja, joiden avulla on mahdollista luoda selainpuolen JavaScript-sovelluksia käyttäen Java-ohjelmointikieltä. Mm. Vaadin-sovelluskehys käyttää GWT-komponentteja. (Google 2008a; Grönroos 2012, 30–33.)

2.4 Vaadin-sovelluskehys

2.4.1 Esittely

Vaadin on suomalaisen IT-millin kehittämä web-sovelluskehys, joka ennen uudelleen brändäystä oli nimeltään IT Mill Toolkit. Työkalujärjestelmän kehityksessä käytettiin Vaatimen versiota 6, jota edeltävä versio julkaistiin nimellä IT Mill Toolkit 5. Opinnäytetyötä kirjoittaessa tällä hetkellä uusin versio kantaa nimeä Vaadin 7. Vaadin on suomea ja tarkoittaa naarasporoa. (Grönroos 2012, 7–8.)

Vaadin mahdollistaa ohjelmoijalle käyttöliittymän toteuttamisen siten, ettei hänen tarvitse käyttää internet-selaimessa toimivia kieliä, kuten HTML-merkkauskieltä ja Ja-

vaScriptiä. Koska valtaosa suosituimmista Internet-selaimista ymmärtää edellä mainittuja kieliä, muttei Javaa, täytyy Vaatimen kääntää Java-koodi Internet-selaimelle sopivaksi. Kun ohjelmoijan ei tarvitse käyttää ohjelmointiin muuta kuin Javaa, muistuttaa Vaadin-sovellusten käyttöliittymien ohjelmointi työpöytäsovellusten kehittämistä Swingin tai AWT:n avulla. (Grönroos 2012, 3)

2.4.2 Arkkitehtuuri lyhyesti

Joistakin Web-sovelluskehyksistä poiketen Vaadin toimii sekä sovelluksen käyttäjän tietokoneella että web-palvelimella. Käyttäjälle selaimessa näkyvä web-käyttöliittymä kommunikoi suhteellisen paljon palvelinpuolen kanssa. Vaadin-sovelluksen pystyy tekemään pitkälti pelkällä Java-ohjelmointikielellä, joskin Vaadin-sovelluksen ulkonäköön voi vaikuttaa paljon käyttämällä Javan lisäksi CSS-tekniikkaa.

Vaatimen selainpuolen moottori (Client-Side Engine) hallinnoi käyttöliittymän esitystä web-selaimessa käyttäen Google Web Toolkit -komponentteja. Selainpuolen moottori kommunikoi käyttäjän vuorovaikutuksesta ja käyttöliittymässä tapahtuneista muutoksista palvelinpuolen päätesovittimen (Terminal Adapter) kanssa. Selainpuolen moottori ja päätesovitin kommunikoivat JSON:iin (JavaScript Object Notation) perustuvalla UIDL-kielellä. Web-palvelimen puolella Vaadin-sovellus toimii Servletinä käyttäen Java-teknologiaa. Vaadin kuitenkin abstrahoi hyvin kaikki käyttämänsä tekniikat ohjelmoijalta näkymättömiksi. (Grönroos 2012, 30.)

2.4.3 Käyttöliittymäkomponentit ja sivujen asettelu

Vaadin sisältää joukon käyttöliittymäkomponentteja, joista useita on käytetty työkalujärjestelmänkin kehityksessä. Työkalujärjestelmässä eniten käytettyjä komponentteja olivat alasvetovalikko (ComboBox), painike (Button), tekstikenttä (TextField) ja lomake (Form).

Sivun asetteluun Java-koodin kautta Vaadin tarjoaa useita erilaisia Layout-luokkia, joiden avulla käyttöliittymäkomponentit voidaan näyttää esimerkiksi päällekkäin ja vie-

rekkäin. Työkalujärjestelmässä on käytetty pääasiassa kolmea eri luokkaa (tai niistä periytyneitä luokkia) komponenttien asetteluun; HorizontalLayout, VerticalLayout ja GridLayout. Edellä mainituista luokista HorizontalLayout asettelee komponentit vierekkäin ja VerticalLayout päällekkäin. GridLayout laittaa komponentit ruudukkoon aloittaen täyttämään sitä vasemmalta oikealle, ylhäältä alas -järjestyksessä. (Grönroos 2012, 156–161.)

2.4.4 Teemat

Työkalujärjestelmässä käytetyn Vaadin-sovelluskehyksen version 6.8.6. mukana tulee kolme sisään rakennettua teemaa (theme), joiden avulla sivujen ulkonäköön ja asetteluun voi vaikuttaa (Vaadin 2012a). Teemat koostuvat mm. CSS-tiedostoista ja kuvista, jotka voi ottaa käyttöön Java-koodissa. Oletusteemojen lisäksi teemoja voi tehdä myös itse, mutta niiden tekemiseen täytyy käyttää CSS-tekniikkaa. (Grönroos 2012, 210.)

2.4.5 Vaatimen lisäosat

Vaatimeen on saatavilla lukuisia lisäosia (add-ons) jotka lisäävät toiminnallisuutta Vaadin-sovelluskehykseen (Vaadin 2012b). Työkalujärjestelmässä on käytetty kahta Vaadin-lisäosaa, CustomFieldiä ja Navigatoria. CustomField-lisäosa on tarkoitettu käytettäväksi Vaatimen Form-komponenteissa. Navigator-lisäosa mahdollistaa joidenkin sivuston käytettävyyttä parantavien ominaisuuksien toteuttamisen.

2.5 Hibernate

2.5.1 Esittely

Kun tietokanta-ajuri on liitetty Java-projektiin, voi tietokannanhallintajärjestelmän kanssa kommunikoida käyttäen Javan JDBC-rajapintaa. Tämän vuoksi järjestelmän voisi toteuttaa halutessaan tekemällä tietokantaoperaatiot suoraan JDBC:n kautta. Tämä tapa tekee olioiden persistoinnin kuitenkin melko hankalaksi ja epäoliomaiseksi, jonka vuoksi on olemassa avoimen lähdekoodin Java-sovelluskehyksiä kuten Hibernate.

Alkuperäinen tapa luokkien ja tietokantataulujen väliseen linkitykseen (ns. mapping) Hibernatessa oli käyttää XML-tiedostoja (.hbm.xml). JDK 5.0:n annotaatioiden myötä on Hibernatessa ollut mahdollista linkittää Java-luokat suoraan tietokantatauluihin ilman XML-tiedostoja, koska annotaatioita pystyi käyttämään XML:n kaltaisesti metadatana. (JBoss 2013).

Sen lisäksi, että annotaatiot ovat all-Java-paradigman kannalta parempi tapa tehdä mapping-määrittelyt, ovat ne myös ylläpidon kannalta huomattavasti parempi ratkaisu, koska Java-luokkien ja tietokantataulujen lisäksi ei tarvitse ylläpitää XML-tiedostoja. Esimerkiksi jos tietokantatauluun linkitettävään Java-luokkaan tehdään muutos, täytyy myös tietokantataulua ja XML-tiedostoa muokata, mikäli käyttää XML-tiedostoja linkitykseen.

2.5.2 HQL

HQL on Hibernaten oma oliopohjainen kyselykieli, mikä mahdollistaa tietokannan hallintajärjestelmästä riippumattomien CRUD-operaatioiden tekemisen. Kaikkia HQL-kielen mahdollistamia lausekkeita (expressions) ei kuitenkaan voi käyttää, jos tietokannanhallintajärjestelmä, kuten esimerkiksi MySQL ei tue niitä. (JBoss 2004.) Hibernate muuttaa HQL-kielen asetuksissa määritellylle MySQL-murteelle.

2.5.3 Criteria API

Criteria API on HQL-kieltä elegantimpi tapa toteuttaa tietokantaoperaatioita. Criteria API mahdollistaa Java-oliosuuntautuneen tavan toteuttaa monimutkaisia kyselyitä tietokannasta. (JBoss 2005.) All-Java-paradigman kannalta Criteria API:n käyttäminen on erinomainen tapa toteuttaa kyselyjä, koska Java-koodiin ei tarvitse sekoittaa merkkijonoina kyselykieliä, kuten SQL- tai HQL-kieliä.

2.5.4 Envers

Envers on versionhallinta, mikä mahdollistaa historiatietojen tallentamisen tietokantaan persistoitavista luokista. Työkalujärjestelmän tapauksessa Enversiä voi käyttää työkalujen eri versioiden tallentamiseen, mikä mahdollistaa työkaluihin tehtyjen eri päivitysten vertaamisen toisiinsa. Hibernaten versiosta 3.5 alkaen Envers on ollut osa Hibernaten ydinmoduulia (ns. core module). (JBoss 2010.)

2.6 Apache Tomcat -palvelin

Tomcat on Apache Software Foundationin kehittämä avoimen lähdekoodin web-palvelin. Tomcat toteuttaa Java Servlet ja JavaServer Pages -määrittelyn tehden palvelimesta sopivan ympäristön mm. Java- ja Servlet-teknologioita käyttävien Vaadin-sovellusten toiminnalle. (The Apache Software Foundation 2013.) Työkalujärjestelmän toteuttamisessa on käytetty Tomcatin versiota 7.

2.7 MySQL

2.7.1 Tietokannanhallintajärjestelmä

MySQL on suosittu avoimen lähdekoodin tietokannanhallintajärjestelmä (MySQL 2013a). Työkalujärjestelmän toteuttamiseen käytetään MySQL:n ilmaista Community

Editionia. Koska työkalujärjestelmässä on tarkoituksena käyttää Hibernate-sovelluskehystä tietojen tallentamiseen tietokantaan, ei ohjelmoijan tarvitse juurikaan välittää siitä, mitä tietokannan hallintajärjestelmää käytetään. Toisaalta Hibernate mahdollistaa natiivien, tietokannanhallintajärjestelmästä riippuvien kyselyjen ja operaatioiden tekemisen MySQL-murteella. Natiiveja kyselyitä käyttäessä erityisesti SQL-standardista poikkeaminen tietokantaoperaatioissa hankaloittaa tietokannanhallintajärjestelmän vaihtamista toiseen.

2.7.2 MySQL Query Browser

Query Browser on graafinen työkalu MySQL-tietokannanhallintajärjestelmän käyttämiseksi (MySQL 2013b, 2). Graafisen käyttöliittymän kautta tietokantaoperaatioiden tekeminen vaatii vähemmän kirjoittamista ja muistamista. Query Browserin kautta on helppoa katsoa luotuja tietokantatauluja ja tietokantaan tallennettua tietoa. Työkalujärjestelmän tapauksessa Query Browseria on käytetty Hibernaten kautta tehtyjen tietokantaoperaatioiden toimivuuden testauksessa.

2.7.3 Ajuri

Koodin ja tietokannanhallintajärjestelmän yhteiskäyttöön tarvitaan ajuri. Projektiin täytyi lisätä Java-sovelluksessa toimiva MySQL-ajuri, koska projektissa käytettävä tietokannanhallintajärjestelmä on MySQL. Työkalujärjestelmän tapauksessa ajurin käyttöön ottamiseksi tarvitsee vain liittää mysql-connector-java-5.1.21-bin.jar-pakkaus projektissa käytettäviin kirjastoihin.

2.8 SimpleCaptcha

Koska työkalujärjestelmä on avoin sivusto, on se altis esimerkiksi bottihyökkäyksille. Työkalujärjestelmän tapauksessa bottihyökkäyksellä voisi taltioida jatkuvasti uutta tietoa joidenkin lomakkeiden kautta kuormittaen sekä palvelinta että täyttäen tietokantaa turhalla tiedolla. SimpleCaptcha on Java-sovelluskehys, joka on tehty kuva- ja äänivar-

mennuksien luomiseen. Työkalujärjestelmässä käytetään vain kuvavarmennusta, mutta SimpleCaptchan pitäisi mahdollistaa myös helpon äänivarmennuksen tekeminen. (SimpleCaptcha 2013.)

3 Toteutus

3.1 Esittely

Toteutus-luvussa on kerrottu lyhyesti projektiin valituista sovelluskehyksistä ja niiden valintaan vaikuttaneista syistä. Myös projektin liikkeellelähdöstä ja projektissa käytetyistä muista työkaluista on kerrottu lyhyesti.

Vaadin-sovellusten selainpuoli ja palvelinpuoli tekevät tiivistä yhteistyötä, joten niistä kirjoittaminen eri luvuissa olisi ollut hankalaa. Tämän takia jaoin toteuttamisesta kertomisen kahteen isompaan kokonaisuuteen. Niistä toisessa kerrotaan Vaatimen soveltamisesta ja toisessa Hibernaten käyttämisestä projektissa. Vaatimen ja Hibernaten välisestä tiedonkulusta on kerrottu luvussa 3.5.6.

3.2 Sovelluskehysten valinta ja all-Java-paradigma

Koska opinnäytetyötä aloittaessani järjestelmän toteuttaminen kokonaan Javalla ei ollut minulla ensisijaisesti mielessäni vaan tavoitteeni oli kehittää itseäni sovelluskehysten ja web-ohjelmoinnin saralla, en pyrkinyt valitsemaan ihanteellisia sovelluskehyksiä all-Java-paradigmaa ajatellen. Teknisesti ottaen halusin kirjoittaa itse mahdollisimman vähän matalan tason koodia ja siihen Vaadin vaikutti olevan erinomainen ratkaisu. Vaatimen mahdollistamat RIA-ominaisuudet vaikuttivat myös riittäviltä työkalujärjestelmän kaltaista tietovarantoa varten. Hibernaten valitsin pitkälti siksi, että halusin saada myös kokemusta jostakin ORM-sovelluskehyksestä. Hibernate vaikutti olevan yksi suosituimmista, ellei suosituin Javalle saatava ORM-ratkaisu. Muita Vaatimen valintaan vaikuttavia tekijöitä oli mm. se, että se on suomalaisen yrityksen kehittämä. Hibernaten valintaan vaikutti paljon myös se, että useissa ohjelmistoalan työnpaikkailmoituksissa

edellytetään kokemusta kyseisen sovelluskehyksen käytöstä. Järjestelmän kehityksen edetessä minulle heräsi mielenkiinto tutkia, kuinka pitkälle järjestelmän voisi kehittää käyttäen pelkkää Javaa.

3.3 Ohjelmistoprojektin aloittaminen

Kehitin järjestelmää käyttäen kehitysympäristönä Eclipse Indigoa ja siihen asennettua Vaadin Plug-iniä. Javalla kehitettävää projektia varten minun piti myös ladata ja asentaa JDK. Lisäksi asensin Eclipseen Apachen Tomcat-palvelimen, jotta sovellusta voisi testata. Edellä mainitut työkalut valikoituivat, koska niiden asentamiseen löytyi hyvät ohjeet Vaatimen Internet-sivuilta. Vaatimen Internet-sivujen mukaan Vaadin ei ole sidottu mihinkään tiettyyn kehitysympäristöön, mutta Eclipsen käyttäjiä suositeltiin asentamaan Vaadin Plugin Eclipseen. Vaadin-liitännäinen sisälsi pelkkien kirjastojen lisäksi muita hyödyllisiä asioita, kuten esimerkiksi Wizardit ja API-dokumentaation. (Grönroos 2012, 6–13.)

Hibernaten käyttöön ottamista varten asensin Eclipseen Hibernate Eclipse Plugin -lisäosan ja latasin useita Hibernaten käytön kannalta oleellisia jar-tiedostoja, jotka sisälsivät tarvittavia kirjastoja. MySQL-tietokannanhallintajärjestelmän otin käyttöön Hibernaten asetusten määrittelytiedostossa (hibernate.cfg.xml). Projektin edetessä havaitsin, että konfiguraatio-tiedostoa tarvitse ainakaan MySQL:n käyttöönottoasetuksien takia, koska ne pystyi määrittelemään suoraan Java-koodin kautta.

Tyypillisesti projekteilla on jokin työnimi ja työkalujärjestelmälle valitsin nimeksi ZombiePrototypeApplication. Nimi muutetaan asiallisemmaksi, jos järjestelmä jossakin vaiheessa viimeistellään ja otetaan käyttöön. Tietokantataulujen luomiseen tarkoitetulle tekemälleni Java-sovellukselle annoin yhtenäisyyden vuoksi nimen ZombieTableGenerator.

3.4 Käyttöliittymän ja toimintojen toteuttaminen soveltamalla Vaadinta

3.4.1 Esittely

Kuten useat muutkin verkkosovellukset, työkalujärjestelmä koostuu erilaisista sivuista, jotka sisältävät useanlaisia komponentteja, kuten painikkeita ja tekstikenttiä. Näin ollen päädyin jakamaan järjestelmän ominaisuuksien toteutuksen kuvauksen kertomalla jokaisen sivun tekemisestä yksi kerrallaan. Niiden ominaisuuksien, joita ei voi luokitella sivuksi, toteuttamisesta kerron Toteutus-osuuden jälkimmäisissä luvuissa. Näitä ominaisuuksia ovat esimerkiksi navigointi ja validointi.

3.4.2 Hakusivu

Toimeksiannossa vaaditut ominaisuudet

Toimeksiannon mukaan järjestelmässä tuli olla hakusivu, jonka kautta erilaisia työkaluja oli mahdollista hakea. Järjestelmän ollessa ensisijaisesti suunnattu sellaisille käyttäjille, joilla ei ole hakemansa työkalun nimeä tiedossa, ei sivulla ollut erityistä tarvetta tekstihakukentälle.

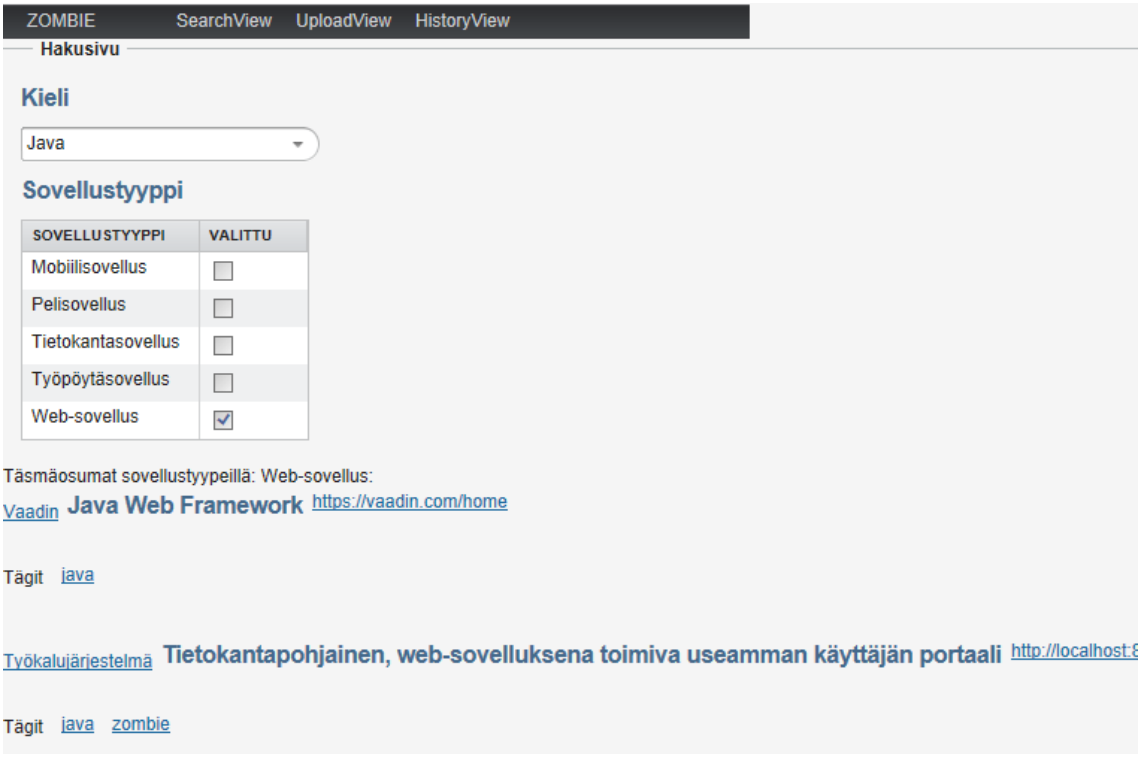
Toimeksiannon mukaan sivuilla tuli olla mahdollisuus hakea sopivaa työkalua ohjelmointikielen ja sovellustyypin perusteella. Haun tekemisen jälkeen järjestelmän tuli listata hakutulokset esimerkiksi työkalujen tietoihin taltioitujen tagien tai muun metadatan perusteella.

Toimeksiannossa esimerkkejä sovellustyypeille oli neljä: mobiilisovellus, tietokantasovellus, työpöytäsovellus ja web-sovellus. Valitsemalla useamman sovellustyypin on haun tarkoitus tarkentua ja pois sulkea sellaiset työkalut, joihin ei ole tallennettu yhtään valituista sovellustyypeistä.

Lomake

Hakusivun (Kuva 4) runkona päädyin käyttämään Vaatimen Form-komponenttia, joka vaikutti toimivalta ratkaisulta tilanteeseen. Lomakkeen yläosaan sijoitin hakuun tarvit-

tavat kentät (ns. fields) kielen ja sovellustyypin valintaa varten. Form-komponentin alatunniste (ns. footer) osa osoittautui hyväksi paikaksi hakutulosten näyttämiseen, koska siihen oli mahdollista sijoittaa muitakin kuin Field-luokasta periytyviä komponentteja, toisin kuin lomakkeen yläosaan.



Kuva 4. Hakusivu-lomake. Kielen ja sovellustyypin valintakentät sekä alatunnisteeseen listatut hakutulokset.

Kielen valintatoiminto

Päädyin toteuttamaan hakusivun ohjelmointikielen valintaominaisuuden käyttäen alasvetovalikkoa. Päädyin tähän ratkaisuun mm. siitä syystä, että erilaisia ohjelmointikieliä on hyvin runsas määrä. Lisäksi valintaruutujen tai valintapainikkeiden avulla toteutettuna kielen valinta -osio olisi voinut vaatia runsaasti sivutilaa ja haitata käytettävyyttä. Esimerkiksi sivua olisi joissakin tilanteissa voinut joutua rullaamaan alaspäin, jotta löytäisi oikean kielivaihtoehdon.

Vaadin ComboBox -komponentti toimii työkalujärjestelmässä pohjana kielen valinta -alasvetovalikolle. Totesin ComboBoxin olevan hyödyllinen komponentti näyttämään suuria määriä vaihtoehtoja, jotka haetaan tässä tapauksessa järjestelmään lisättyjen kielten listasta. Tiedot listaan haetaan tietokannasta, josta enemmän on kerrottu osiossa 3.5.

Kielen perusteella tehtäviä hakuja sujuvoittaakseni liitin kielen valinta-alasvetovalikkoon kuuntelijan (ns. listener), jolloin joka kerta, kun alasvetovalikosta valittiin jokin vaihtoehto, tapahtui haku. Näin ollen erilliselle Hae-painikkeelle ei ollut tarvetta kielenvalintatoiminnon osalta. Haun tulokset määräytyvät kielen valinnan lisäksi sen mukaan, mitä sovellustyyppejä on valittu kielivalinnan alla olevasta sovellustyyppitaulukosta

Sovellustyypin valintatoiminto

Erilaisia sovellustyyppivaihtoehtoja on suhteellisen vähän, joten mm. siksi päädyin niiden valitsemistoiminnossa käyttämään monivalintaruutuja. Toinen syy monivalintaruuduille on se, että haun tarkentamista varten tuli olla mahdollista valita useampi vaihtoehto. Valintaruudut laitoin kaksisarakkeiseen taulukkoon, jonka toteutin Vaatimen Table-komponenttia hyväksi käyttäen. Kuten kielen valinnankin tapauksessa tapahtuu haku joka kerta, kun jotakin valintaruutua klikkaa hiirellä.

Hakutulosten toiminnot

Haun tekemisen hakutulokset listautuvat hakusivun alatunnisteeseen allekkain. Kunkin listatun työkalun tiedoissa näkyy työkalun nimi, kuvaus ja linkki työkalun Internet-sivuille. Päädyin toteuttamaan linkin painamistoiminnon siten, että Internet-selain siirtyy työkalujärjestelmän sivusta työkalun tietoihin tallennetulle sivulle.

Mikäli työkalulle on tallennettu tageja, tulevat ne näkyviin muiden työkalun tietojen alapuolelle. Klikkaamalla tagia tapahtuu uusi haku tagin perusteella. Takaisin pääsy edellisiin tuloksiin ei kuitenkaan onnistu painamalla selaimen takaisin-painiketta, mikä voi vaikuttaa hieman epäjohdonmukaiselta ja huonolta käytettävyyden kannalta. Syy tämänlaiseen toimintaan on Navigator-lisäosan käyttöönotto järjestelmän kehityksen ollessa jo melko pitkällä. Ongelma oli mielestäni melko pieni siinä mielessä, että edelliselle sivulle pääsee tekemällä uuden haun samoilla ehdoilla kuin aiemminkin. Näin ollen päädyin keskittymään järjestelmän kehityksessä oleellisempiin asioihin. Navigator-lisäosan tuomista haasteista on kerrottu lisää kohdassa 3.4.7.

Hakutulosten asettelu

Form-komponentin alatunnisteessa voi esittää komponentteja melko mielivaltaisesti, koska alatunnisteen asettelua on helppoa säätää mm. Vaatimen HorizontalLayout Verti-

calLayout ja GridLayout -luokkia käyttäen. Kuten kuvassa 5 näkyy, toteutin hakutulosten listaustoiminnon niin, että jokainen eri työkalu tietoineen on omalla rivillään ja jokaisen työkalun sisältämät perustiedot ovat vierekkäin. Lisäksi jokaisen työkalun mahdolliset tagit on listattu kunkin työkalun alle.

Teknisesti ottaen esimerkiksi kuvassa 5 näkyvän Xampp-työkalun perustiedot eli nimi, kuvaus ja linkki on lisätty vierekkäin HorizontalLayout-olioon. Kyseisen työkalun tagit on sijoitettu GridLayout-olioon, joka tarvittaessa rivittää tagit useammalle riville, jos niitä on sivusuunnassa paljon. Sekä Xampp-työkalun perustiedot sisältävä HorizontalLayout-olio ja tagit sisältävä GridLayout-olio on sijoitettu VerticalLayout-olioon, kuten myös Vaatimen ja työkalujärjestelmän tapauksessa. Tällöin työkalujen tiedot tageineen näkyvät eri riveillä.

Kuva 5. Hakutulosten asettelu.

3.4.3 Työkalun tietojen tallennus- ja päivityssivut

Toimeksianto ja toteuttaminen karkealla tasolla

Toimeksiannossa ei ollut erityisen tarkkaan määritetty, miten työkalujärjestelmän tietojen tallentaminen ja päivittäminen tulisi hoitaa, joten toteutustapa oli melko vapaa. Tallennettavan työkalun tietoihin piti jollakin tapaa olla mahdollista syöttää tieto sen kielestä ja sovellustyypistä. Muussa tapauksessa haku kyseisten tietojen perusteella olisi

ollut mahdotonta. Koska hakutuloksien listauksessa näkyvät työkalun nimi, kuvaus ja Internet-osoite, täytyi ko. tietojen taltioimista varten olla myös jonkinlaiset kentät.

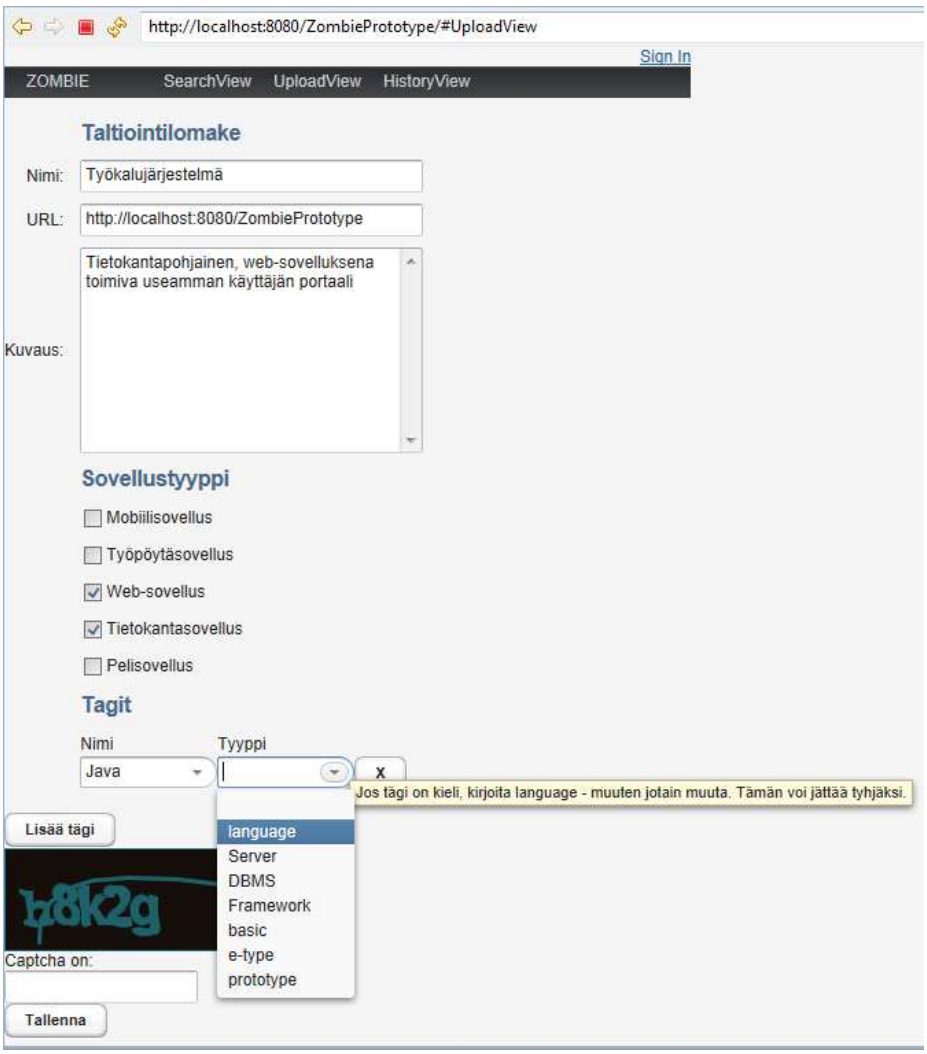
Tein tallennus- ja päivityssivuista lähes identtisen näköiset, mikä helpotti ohjelmointityötä, koska pystyin tekemään molemmille sivuille yhteisen rungon, josta kumpikin periytyvät. Näin ollen sivujen toteuttamisesta on kerrottu yhtenä komponenttina tuoden tallennus- ja päivityssivujen väliset erot esille tämän osion lopuksi. Työkalujärjestelmän kehityksen aikana toimeksiantajan kanssa käytyjen keskustelujen yhteydessä pohdimme, että tallennus- ja päivityssivuilla olisi hyvä olla myös kuvavarmennus.

Sivun runko

Web-lomakkeet ovat tietojen tallentamiseen ja muokkaamiseen varsin käteviä ja selkeitä, joten päädyin käyttämään Vaatimen Form-komponenttia BaseForm-nimisen luokan pohjana. BaseForm puolestaan toimii pohjana sekä taltiointi- että päivityssivuille. Runkoa toteuttaessa lisäsin tekstin syöttöön soveltuvat kentät sekä työkalun nimeä, kuvausta ja Internet-osoitetta varten. Lisäksi tein sovellustyyppitiedon lisäämistä varten monivalintaruudut, joita käyttäen on mahdollista tallentaa työkalulle halutessaan useita sovellustyyppejä.

Koska erilaisia ohjelmointikieliä on lukuisia ja kaikkien järjestelmässä tarvittavien kielten määrä ei ole ennakkoon tiedossa, tein kielen lisäämisen tietoihin tagien kautta. Tageille päädyin toteuttamaan oman komponentin nimeltä TagTypeCombo (kuva 6), jossa on kaksi automaattisella täydennyksellä varustettua kenttää, joista toinen on tagien nimeä ja toinen tyyppiä varten.

Kun tagin tyypiksi kirjoitetaan tai valitaan "language", tallentuu tagin nimikentässä oleva tieto ohjelmointikieleksi työkalujärjestelmään, jos sitä ei ennestään siellä ole. Tällöin tallennetun ohjelmointikielen avulla voi jatkossa hakea työkaluja hakusivulta käsin. Koska tagin lisääminen kieleksi on prototyypin tekijänkin mielestä hieman sekava toiminnallisuus, päädyin lisäämään myös TagTypeComboon työkaluvihjeen tagin tyyppikenttään (kuva 6).



Kuva 6. Työkalun Taltiointilomake.

CustomField-lisäosa

Kun tein ensimmäistä tagien lisäämistä varten tehtyä komponenttia, joka koostui HorizontalLayout-olioon lisätyistä kahdesta ComboBox- sekä Button-oliosta vierekkäin, törmäsin ongelmaan. Vaatimen Form-komponentin ylempi osa ei sallinut muun tyyppisiä olioita kuin Field-luokasta periytyviä. Tähän ongelmaan kuitenkin löysin prototyypille riittävän toimivan CustomField-lisäosan. Sen avulla pystyi tekemään Field-tyyppisen komponentin, joka koostuu muista komponenteista.

Pienenä takaiskuna koin CustomFieldiä soveltaen tehdyn komponentin käyttämisessä sen, etten löytänyt kunnollista tapaa saada kyseiseen kenttään toimivaa validointia toisin kuin normaaleissa Field-tyyppisissä komponenteissa.

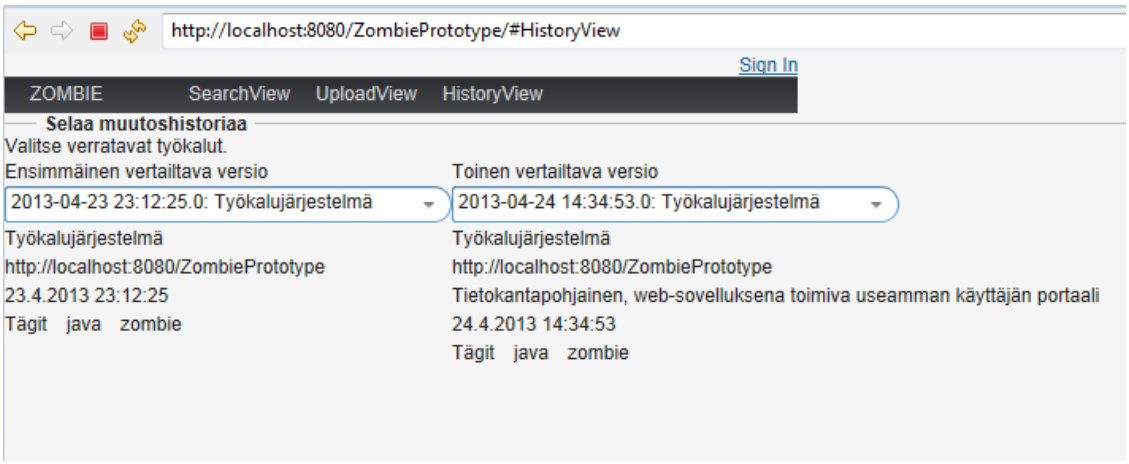
Kuvavarmennus

Tietoturvan kannalta työkalujärjestelmässäkin todettiin hyödylliseksi käyttää kuvavarmennusta esimerkiksi mahdollisten bottihyökkäysten estämiseksi. Kuvavarmennuksen soveltamisesta työkalujärjestelmässä on kerrottu tarkemmin kohdassa 3.4.6.

3.4.4 Historiasivu

Historiatietojen pääasiallinen tarkoitus on ainoastaan näyttää, millaisia muokkauksia eri työkalujen tietoihin on tehty. Syy tähän on lähinnä käyttäjien mahdollisuus katsoa, onko työkalujen tietoja muokattu jossakin vaiheessa ja miten.

Toteutin taltioitujen työkalujen historiatietojen katselemisen siten, että kun käyttäjä ollessaan jonkin työkalun päivityssivulla klikkaa historiasivun valintaa yläpalkista, avautuu historiasivu (kuva 7). Historiasivulle ilmestyviin alasvetovalikoihin listautuu muokatun työkalun eri versioiden nimet. Jokaista muokkausta edeltää muokkaushetken päivämäärä ja aika. Mikäli historiasivulle ei navigoi päivityssivun kautta, ei sivulle myöskään ilmesty alasvetovalikoita vaan teksti, jossa neuvotaan miten historiatietoja pääsee katsomaan.



Kuva 7. Historiasivu.

Historiasivun toteuttamiseen käytin suhteellisen vähän aikaa, joten joitakin käyttäjiä saattaa sivussa kaihertaa se, että alasvetovalikoihin tulee näkyviin kustakin työkalusta jokainen päivitys, vaikkei työkalun tietoja olisi edes muokattu. Lisäksi molempiin alas-

vetovalikoihin listautuvat kaikki versiot työkaluista, mikä mahdollistaa myös työkalun saman version vertaamisen itseensä.

3.4.5 Lomakkeiden kautta lähetetyn tiedon validointi

Tarve validoinnille

Työkalujärjestelmän kehittämisen alkuvaiheessa tietoturvan kannalta tarve validoinnille vaikutti selkeältä. Yhtenä syynä tähän oli se, että aluksi olin käyttänyt järjestelmän kehittämiseen pääasiassa HQL-kieltä. HQL mahdollistaa HQL-injektiot varsinkin silloin, kun ei käytä valmisteltuja lausekkeita (ns. prepared statements) HQL-operaatioiden seassa. Kun myöhemmässä vaiheessa projektia korvasin HQL-lausekkeet Criteria API:n metodeilla, validoinnin tarve tietoturvan kannalta väheni ymmärtääkseni merkittävästi. Käsittääkseni Criteria API käyttää valmisteltuja lausekkeita ns. "pinnan alla" injektioiden estämiseksi.

Criteria API:n käyttöön otto ei kuitenkaan sulje pois tarvetta validoinnille, koska validoinnista voi olla hyötyä esimerkiksi palautteen antamisessa käyttäjälle web-lomakkeita täyttäessä ja lähettäessä. Vaatimen Form-komponenttiin lisättyihin kenttiin on mahdollista kiinnittää validaattori, joka antaa palautetta huonosta syötteestä (Grönroos 2012, 141). Työkalujärjestelmän lomakkeissa on tarkoituksena olla mahdollista tallentaa ainakin työkalun nimi, nettiosoite ja lyhyt kuvaus työkaluista. Siksi katsoin välittömän palautteen antamisen hyödylliseksi kyseisissä kentissä. Lisäksi olisin halunnut toimivan validaattorin myös CustomField-lisäosalla tehdylle tagien syöttöön tarkoitetulle TagTypeCombo-kentälle, mutta en löytänyt siihen toimivaa ratkaisua.

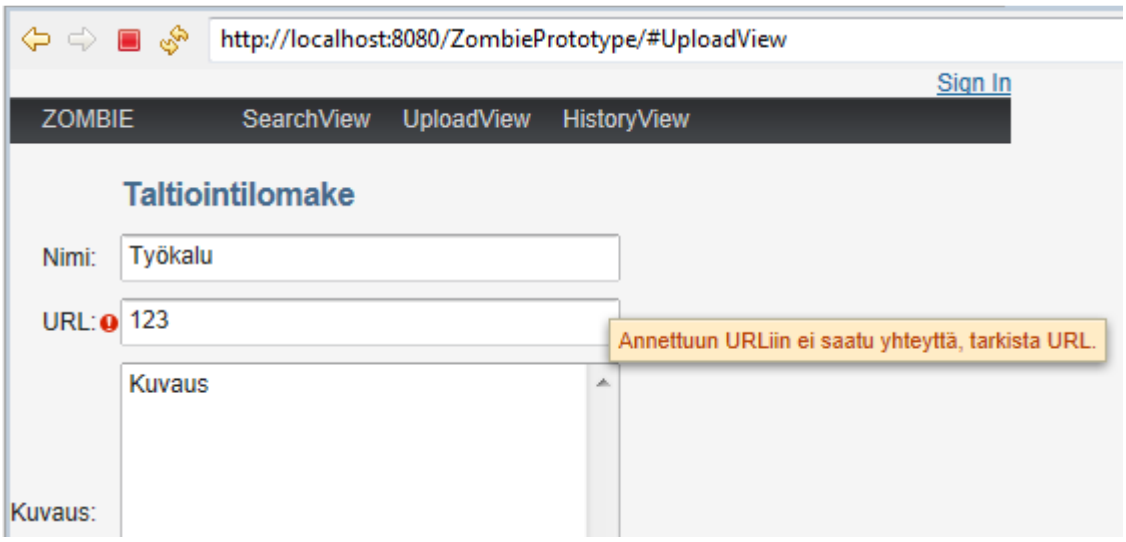
Validoinnin liittäminen kenttiin

Vaadin-sovelluskehys tarjoaa itsessään useita hyödyllisiä kirjastoja validointiin, kuten RegexpValidator-luokan ja Validator-rajapinnan, jotka mahdollistavat helpon tavan liittää validointi Form-komponenttiin. RegexpValidator-luokan käyttäminen olisi ollut hyvin käytännöllinen tapa määrittää sallitut merkit ja merkkijonojen pituudet tekstikenttiin. Sen käyttämät säännölliset lausekkeet (ns. regular expressions) kuitenkin omaavat oman syntaksinsa, mitä voisi pitää all-Java-paradigman vastaisena. Tästä syystä päädyin toteuttamaan hieman erilaisen validaattorin, joka kuitenkin käytti pohjana Vaatimen

omia validointiin tarkoitettuja luokkia erityisesti siksi, että ne pystyi sitomaan lomakkeen kenttiin.

Tekemiäni Validator-rajapinnan toteuttavia luokkia hyödyntäen toteutin validoinnin lomakkeiden tekstikenttiin, joihin on tarkoitettu syötettävän taltioitavan työkalun nimi ja kuvaus. Toisin kuin RegexpValidator-luokaa käyttäen, toteutin kenttiin syötettävien sallittujen merkkien määrittämisen käyttäen Java-algoritmeja.

Internet-osoitekentän tapauksessa annetun tiedon validointi onnistui hieman helpommin, koska sen pystyi toteuttamaan siten, että järjestelmä varmistaa osoitteen oikeellisuuden muodostamalla yhteyden Internet-osoitteeseen. Jos osoitteeseen ei saa yhteyttä, validaattori katsoo annetun tiedon vääräksi. Huono puoli tässä ratkaisussa on mm. se, että jokin järjestelmän käyttäjä voi tahallaan taltioida järjestelmään toimivan haitallisen sivuston osoitteen. Myös osoitekenttään merkatun sivuston hetkellinen alhaalla olo voi hankaloittaa työkalun tietojen tallentamista, koska onnistunut yhteyden saaminen annetulle verkkosivulle vaaditaan.



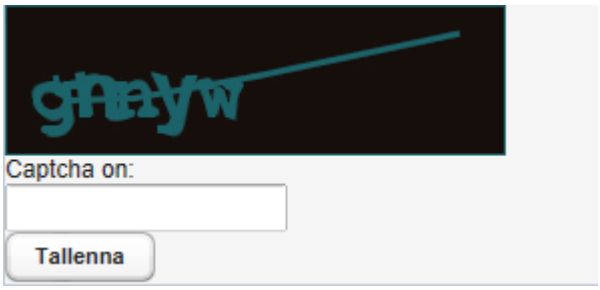
Kuva 8. Virheilmoitus väärin syötetystä tiedosta (hiiren osoitin ei näy kuvankaappauksessa).

Vaadin mahdollistaa välittömän palautteen lomakkeen kenttiin, jos kenttään syötetty tieto on väärää. Käytettävyyden kannalta kyseinen ominaisuus vaikutti hyvältä idealta, joten päätin sen toteuttaa. Virheen kuvauksen saa ilmestymään työkaluvihjettä muistuttavassa ilmoituksessa (Kuva 8), kun hiirellä osoittaa kenttää, johon on kirjoitettu epäkorrektia tietoa.

3.4.6 Kuvavarmenteen toteuttaminen

Sopivan työkalun löytäminen

Kuvavarmenteena (kuva 9) päädyin käyttämään SimpleCaptcha-nimistä työkalua, johon päädyin tiedonhankinnan perusteella. Jälkeenpäin löysin jcaptchafield-nimisen Vaadin-lisäosan, jota en jostakin syystä ollut aiemmin huomannut. En kuitenkaan enää tarvinnut sitä siinä vaiheessa, kun olin saanut SimpleCaptchan toimimaan. En myöskään voi kokeilematta olla varma, saisinko sitä yhtään helpommin toimimaan tarpeideni mukaisesti. SimpleCaptchan toimimaan saamisen jälkeen olin varsin tyytyväinen sen toimintaan.



Kuva 9. Kuvavarmenne ja tekstikenttä johon syötetään kuvassa näkyvä teksti.

Soveltaminen

SimpleCaptcha-sovelluskehyksen käyttöön ottaminen sujui melko kivuttomasti, joskin jouduin tekemään hieman enemmän koodaustyötä kuin olin ajatellut. SimpleCaptchan kotisivuilta ja muualta Internetistä en löytänyt esimerkkejä sen käytöstä Vaatimen kanssa. SimpleCaptchan kotisivuilta löysin esimerkkejä käyttöönotosta JSF-sovelluksessa ja javacodegeeks.com-sivustolta löysin tietoa SimpleCaptchan soveltamisesta GWT-sovelluksessa, mitkä auttoivat SimpleCaptchan käyttämisen ymmärtämisessä (Tsagklis 2010).

Kuvavarmenteen näyttämiseksi työkalujärjestelmässä tein Captcher-nimisen Java-luokan, joka perii ominaisuudet Vaatimen Label-luokalta. Totesin Labelin sopivaksi pohjaksi kuvavarmenteen näyttämiseen, koska se mahdollisti muistiin ladatun kuvan näyttämisen kätevästi. Captcher-luokassa loin olion SimpleCaptchan Captcha-luokasta, jota hyödyntäen sain kuvavarmenteen toteutettua toivomallani tavalla.

Ongelmat SimpleCaptchan soveltamisessa

SimpleCaptchan soveltamisessa työkalujärjestelmään suurimmaksi ongelmaksi muodostui Internet-selainten, kuten Mozilla Firefoxin ominaisuus tallentaa samassa Internet-osoitteessa sijaitsevat samannimiset kuvat ja tiedostot selaimen välimuistiin (Mozilla 2011). Kyseisen toiminnon tarkoitus on parantaa Internet-sivujen lataamisnopeutta käyttämällä välimuistiin tallennettua kuvaa sen sijaan, että kuva ladattaisiin uudelleen. Se toimii järkevästi, kun kuva näyttää samalta mutta kuvavarmenteen tapauksessa SimpleCaptcha luo samannimisen kuvan, joka on kuitenkin aina erilainen. Internet-selaimet eivät kuitenkaan usein ymmärrä kuvan muuttumista, joten ne lataavat vanhan kuvan aiheuttaen sen, ettei SimpleCaptchan luoma vastaus vastaa kuvassa näkyvää tekstiä.

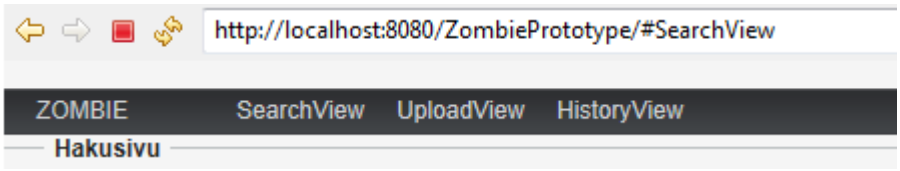
Välimuistin tyhjentämisen vaatimista työkalujärjestelmän käyttäjältä vaikutti huonolta idealta, joten päädyin kiertämään ongelman toteuttamalla Captcheriin toiminnon, jossa jokainen varmennekuva tallentuu eri nimellä. Näin ollen välimuistista ei pitäisi löytyä samannimistä kuvaa. Jotta kuvien nimet eivät sattuisi olemaan samoja, ohjelmoin nimen muodostustoiminnon Captcheriin siten, että jokaisen kuvan nimi päättyy palvelimen sen hetkiseen aikaan mitattuna millisekunneissa. Jokaisen kuvan etuliitteeksi Captcher lisää merkkijonon "img", jonka perusteella Captcher erottaa varmennekuvat muista tiedostoista. Näin ollen Captcher estää palvelimen täyttymisen turhista kuvista poistamalla kaikki img-alkuiset kuvat siitä kansiosta, minne uusi kuva ilmestyy ennen kuin uusi kuvavarmenne luodaan.

Kuvavarmenteen kuvan sijainti

Prototyypissä päädyin siihen ratkaisuun, että varmennuskuva tallennetaan työkalujärjestelmässä valitun teeman sisältämään kansioon. Teemakansiot eivät välttämättä ole ihanteellisia kuvien tallennuspaikkoja, mutta niistä kuvan lataaminen onnistuu melko kätevästi Vaatimen ThemeResource-olion avulla. Tämän vuoksi päädyin tallentamaan kuvat kyseiseen sijaintiin. Suurimpana ongelmana tässä ratkaisussa on se, että jos esimerkiksi järjestelmän tyypillisessä teemojen sijainnissa WebContent/VAADIN/themes ei ole työkalujärjestelmän käyttämää teemaa, ohjelmisto ei toimi kunnolla. Suorituskyvyn kannalta on kuitenkin hyödyllistä pitää Vaatimen sisäänrakennetutkin teemat kyseisessä sijainnissa, joten teeman sijaintikansion ei pitäisi olla ongelma säilyttää varmennuskuvia. (Grönroos 2012, 203.)

3.4.7 Navigoinnin toteuttaminen

Kuten lähes kaikilla Internet-sivuilla, on työkalujärjestelmässäkin hyvä olla mahdollisuus navigoida jotenkin sivuston eri sivuille. Navigoinnin helpottamiseksi käytin Vaatimen tilanteeseen sopivaa valikko-komponenttia MenuBaria, jonka kautta järjestelmän eri sivuille on määrä mennä. MenuBar-komponenttiin lisäsin navigointivaihtoehdoiksi järjestelmän hakusivun, työkalujen lisäyssivun ja historiasivun (kuva 10).



Kuva 10. Työkalujärjestelmän navigointipalkki.

Internet-selainten ja JavaScriptillä tehtyjen sivujen tyypillinen puute interaktiossa näkyy usein esimerkiksi siten, ettei selain osaa päätellä milloin JavaScript-sovelluksessa on siirrytty eri sivulle. Tämän vuoksi monesti selaimen Edellinen sivu- ja Seuraava sivu -painikkeet eivät toimi halutusti web-sovelluksessa. Internet-selaimet toimivat usein hyvin silloin, kun eri sivut ovat eri tiedostoja ja selain pystyy päättelemään sivun vaihtamisen siitä, että selaimeen on aukaistu esimerkiksi eri html-tiedosto. Etsittyäni sopivaa ratkaisua ongelmaani löysin Vaatimen-lisäosan nimeltä Navigator. Navigatorin käyttöön ottaminen sai ainakin selaimen Edellinen sivu -painikkeen toimimaan haluamallani tavalla.

Päädyin ottamaan Navigator-lisäosan käyttöön prototyypin ollessa varsin pitkälle kehitetty, joten oli odotettavissa, että koodia pitää kirjoittaa melko paljon uusiksi. Navigatorin suosima Lazy initialization -tekniikka toi uusia haasteita ohjelmiston muuttamiseen, koska sen toimintamalli oli lähes päinvastainen aiemmin ohjelmoimaani tapaan.

Ennen Navigatoria olin kehittänyt eri työkalujärjestelmän sivujen lataustoiminnon siten, että jokaisen näkymän lomake latautuu uudelleen komponentteineen ja hakee tuoreimmat tiedot tietokannasta. Navigatorin toimintafilosofia on suorituskyvyltään optimaalisempi siinä mielessä, etteivät lomakkeet lataudu kuin kerran ohjelman käytön alkaessa. Se taas toisaalta tarkoittaa sitä, ettei järjestelmässä välttämättä ole uusimpia tietoja haet-

tuna tietokannasta, jos jokin käyttäjä on lisännyt niitä käytön aikana. Tämän ongelman korjatakseni jouduin tekemään melko paljon muutoksia koodiin.

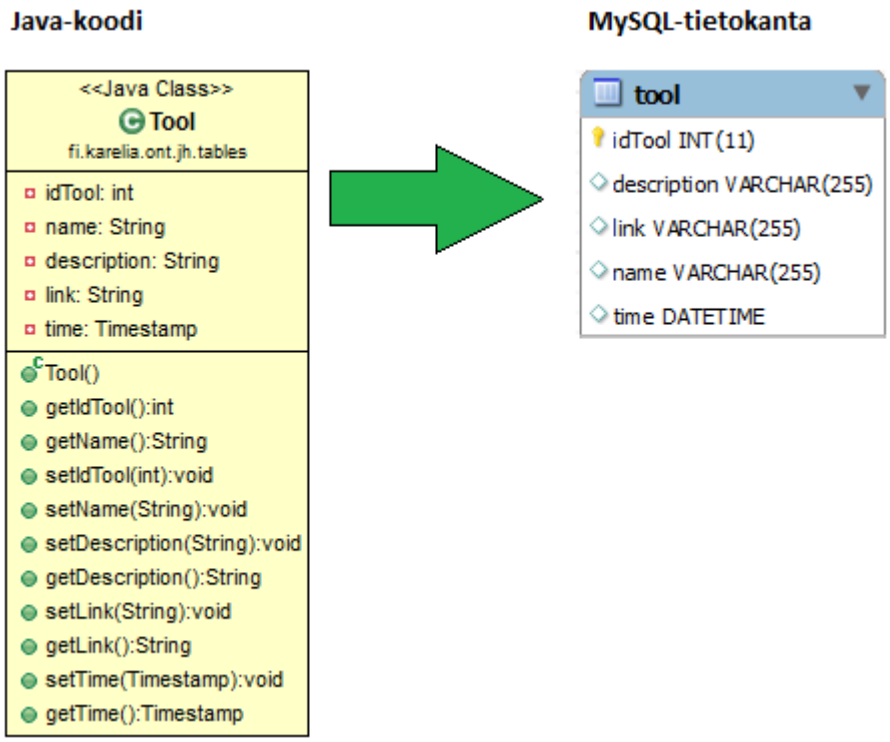
3.5 Tietokantaoperaatioiden toteuttaminen soveltamalla Hibernatea

3.5.1 Tietokantataulujen luominen Java-luokista

Järjestelmän kehityksen aikana havaitsin, että Hibernate mahdollistaa tietokantataulujen luomisen suoraan Java-luokista annotaatioita käyttäen. Tämä oli all-Java-paradigman kannalta hyödyllistä, koska pystyin siten karsimaan valtaosan XML-kielestä projektissa. Alun perin Hibernatessa olioiden persistointi tietokantaan onnistui vain linkittämällä Java-luokat XML-tiedostoihin, jotka linkitettiin tietokantatauluihin. XML-tiedostoista eroon pääseminen helpottaa myös ohjelmiston ylläpitoa, koska annotaatioiden käytön myötä muutoksia tarvitsee tehdä vain Java-luokkiin.

Luodakseni tietokantatauluja Java-luokista tein erillisen Java-sovelluksen. Kyseiselle ohjelmaa käytin silloin, kun tein persistoitaviin Java-luokkiin muutoksia tai tarvitsin kokonaan uusia luokkia. Ohjelman kehittäminen oli varsin yksinkertaista sillä kaiken oleellisen tiedon löysin youtube.com-sivustolla olevalta videolta. (Washington. 2010.)

Työkalujärjestelmässä on kaikkiaan kuusi Java-luokkaa, joista tehdään tietokantataulut suorittamalla ZombieTableGenerator-sovellus. Kyseisiin luokkiin lisäsin @Entity-annotaation luokan määrittelyn eteen, jotta ZombieTableGenerator osaa tehdä luokista tietokantataulut. Lisäksi ZombieTableGenerator luo historiatietojen tallentamista varten tietokantataulut niistä luokista, joissa on @Audited-annotaatio. Historiatietoja varten luoduista tauluista on kerrottu lisää luvussa 3.5.5. Kuvassa 11 näkyy miltä Java-luokasta luotu tietokantataulu näyttää.



Kuva 11. Java-luokka ja siitä luotu tietokantataulu.

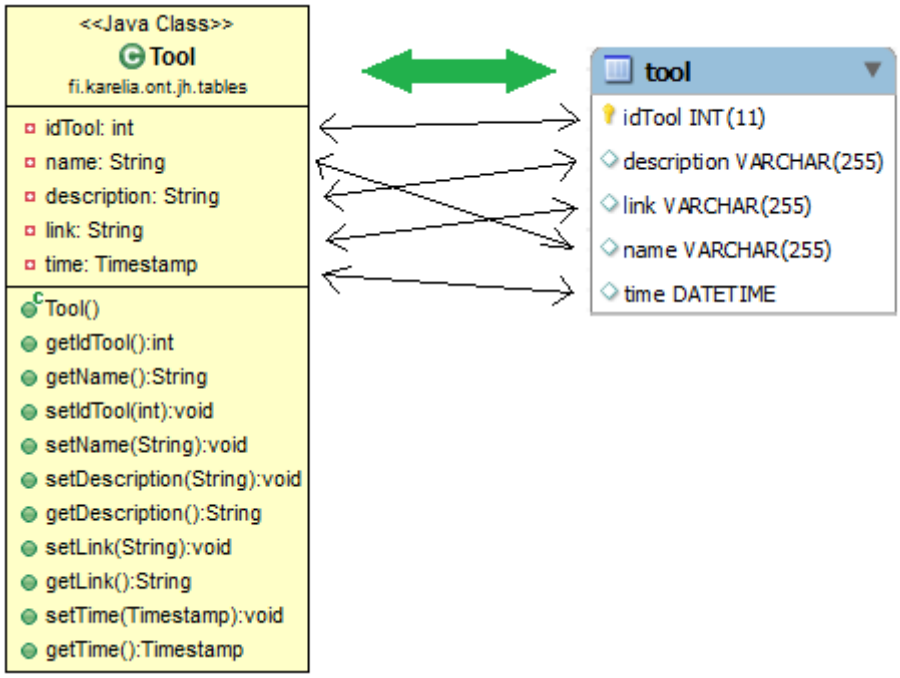
3.5.2 Hibernaten käyttöönotto Java-koodissa

Selkeyden vuoksi tein rajapinnaksi tietokantaoperaatioiden ja muun koodin välille Database-nimisen Java-luokan, jossa toteutin Hibernatea käyttävät julkiset metodit työkalujärjestelmässä. Tämän vuoksi jokaisen työkalujärjestelmän metodin, jonka piti käyttää hyväkseen tietokantaa, täytyi viitata Database-olioon.

3.5.3 Olioiden persistointi tietokantaan

Persistoinnilla tarkoitetaan ohjelman ajon aikaisen olion tilan taltioimista tietokantaan. Yksinkertaistettuna puhutaan tietojen tallentamisesta. Esimerkiksi, kun työkalujärjestelmän työkalun lisäämislomakkeelle kirjoitetaan tiedot niille tarkoitettuihin kenttiin ja painetaan Tallenna-painiketta, tallentuu työkalun tietojen varastointiin tarkoitettu olio (Tool) tietokantaan. Samaiset tiedot on myöhemmin noudettavissa esimerkiksi työkalujärjestelmän hakusivun kautta.

Hibernatea käyttäen olioiden persistoiminen tietokantaan osoittautui melko helpoksi operaatioksi. Tietokantaan tallennettavia olioita olivat käytännössä tietokantatauluihin linkitetyistä luokista tehdyt oliot koodissa. Kyseisiä luokkia ovat työkalujärjestelmässä AppType, Language, Tag, Tool, ToolAppType ja ToolTag. Kaikkien edellä mainittujen luokkien muuttujille, kuten esim. Tool-luokan muuttujille löytyy vastaavat ominaisuudet luokkaan linkitetystä tietokantataulusta (kuva 12).



Kuva 12. Tool-luokasta luodulla tool-tietokantataululla on Tool-luokan muuttujia vastaavat ominaisuudet.

3.5.4 Monimutkaisempien operaatioiden toteuttaminen soveltamalla Criteria API:a

Projektin alkuvaiheessa käytin monimutkaisten tietokantaoperaatioiden tekemiseen pääasiassa Hibernaten HQL-kieltä. Välillä kuitenkin ajauduin tilanteisiin, joissa en vähäisen HQL-tietämyksen takia osannut toteuttaa joitakin tietokantaoperaatioita muuten, kuin MySQL:n SQL-murretta käyttäen. Kokemuksen karttuessa ja projektin edetessä sain käännettyä SQL-kieltä käyttävät metodit HQL-kieltä käyttäviksi, jolloin saavutin yhden tavoitteistani eli tietokannan hallintajärjestelmäriippumattomuuden järjestelmään.

Tietokantaoperaatiot olivat kuitenkin vielä pitkälti all-Java-paradigman vastaisia, sillä HQL on SQL:n tavoin kieli ja sen sekoittaminen Java-koodiin merkkijonojen muodossa ei anna järjestelmästä sitä vaikutelmaa, että se olisi tehty koodaten Javalla. Hibernate-tutoriaaleja lukiessani löysin kuitenkin tietoa Criteria API:sta, joka herätti kiinnostukseni. Havaitsin, että Criteria API:a käyttäen pystyisin korvaamaan myös kaiken HQL-kielen projektista, mikä säilyttäisi kuitenkin riippumattomuuden tietokannanhallintajärjestelmästä ja tekisi Java-koodista vapaan muista kielistä. Koska aikaa oli järjestelmän kehittämistä varten rajatusti, en ehtinyt perehtyä Criteria API:n opettelemiseen niin hyvin kuin olisin halunnut. Näin ollen koodista HQL-kieltä käyttävien metodien muuttaminen Criteria API:a käyttäväksi luultavasti hidasti järjestelmän toimintaa jonkin verran.

3.5.5 Enversin soveltaminen historiatietojen tallentamisessa

Työkalujen historiatietojen toteuttaminen oli ominaisuus, joka ei kuulunut järjestelmältä vaadittuihin ydintoiminnallisuuksiin. Se olisi jäänyt toteuttamatta aikataulullisista syistä, mikäli olisin joutunut toteuttamaan toiminnallisuuden itse tai ilman sopivia apuvälineitä. Lyhyehkölllä tiedonhankinnalla löysin kuitenkin tietoa projektista nimeltä Envers, joka sattumalta kuului käyttämäni Hibernaten version ydinmoduuliin (ns. core module). Hibernaten versiosta 3.5 lähtien Envers on sisällytetty Hibernaten jakeluun, joten en joutunut tekemään ylimääräisiä asennuksia sen käyttöön ottamiseksi.

Lyhyehkön perehtymisen jälkeen Envers-kirjastot vaikuttivat soveltuvan järjestelmään kaikin puolin, myös all-Java-paradigman kannalta. Enversin käyttöönotto ei vaatinut myöskään suuria muutoksia ennestään olevaan koodiin vaan minulle riitti, kun lisäsin @Audited-annotaation jokaiseen Java-luokkaan, joita tarvittiin historiatietojen tallentamiseen. Lisäksi lisäsin ZombieTableGenerator-sovellukseen Enversin toiminnon, joka luo _aud-päätteiset tietokantataulut kaikille luokille, joihin on lisätty edellä mainittu annotaatio. Esimerkiksi Tool-luokalle sovellus loi tool-taulun lisäksi myös tool_aud-taulun, jota tarvitaan historiatietojen tallennukseen. Ohjelma loi myös revinfo-taulun, jota Envers myös tarvitsee toimiakseen oikein.

Prototyyppiin tein ainoastaan työkalujen historiatietojen tallennusominaisuuden ja esimerkiksi kielten historian tallentamista en toteuttanut. Mahdollinen jatkokehittäjä voi kuitenkin helposti ottaa mallia työkalujen historiatietojen tallennusominaisuudesta ja toteuttaa historiatoiminnot myös esimerkiksi kielille. Historiasivuille työkalun tagien hakua toteuttaessa päädyin sellaiseen tekniseen ratkaisuun, ettei järjestelmään tallennettuja työkaluihin linkitettäviä tageja poisteta missään metodissa. Tällöin järjestelmässä vältytään tietynlaisilta viite-eheyteen liittyviltä ongelmilta. Tämän vuoksi työkalujärjestelmässä ainakin toistaiseksi kaikki tallennetut tagit jäävät tallennetuksi tietokantaan.

Pienenä ongelmana pidin sitä, etten löytänyt tapaa ottaa Enversin tietokantaoperaatioiden kuuntelijaluokkia käyttöön suoraan Java-koodista vaan jouduin käyttämään Hibernaten kokoonpanotiedostoa hibernate.cfg.xml vain kyseisten ominaisuuksien takia. Näin ollen Hibernaten kokoonpanoasetukset eivät tulleet tehdyksi Java-koodin kautta. Tiedon etsinnän perusteella sain sen käsityksen, että Hibernaten versiosta 4 alkaen kyseistä ongelmaa ei enää ole. En kuitenkaan enää kesken projektin halunnut lähteä tekemään niin suurta muutosta, että olisin vaihtanut Hibernaten versiota uudempaan jo sen takia, että olisin ymmärtääkseni joutunut koodaamaan joitakin kohtia uudelleen koodista.

3.5.6 Tiedonkulku Hibernaten ja Vaatimen välillä

Työkalujärjestelmässä, kuten monissa muissakin tietokantapohjaisissa tietojärjestelmissä, täytyy useissa tilanteissa välittää tietoa käyttöliittymältä tietokannalle ja päinvastoin. Työkalujärjestelmässä tiedon välitykseen tietokannalle käytin useissa tilanteissa Javan standardikirjastoista löytyviä tietorakenteita, kuten listoja (Lists) sekä taulukkolistoja (ArrayLists). Tietorakenteisiin työkalujärjestelmä lisää persistoitavat oliot ja välittää ne, tilanteesta riippuen, jollekin Database-luokan metodille prosessoitavaksi.

Tarvetta tiedon hakemiseen tietokannasta oli jokaisella työkalujärjestelmän sivulla. Esimerkiksi hakusivun kielen valinta -alasvetovalikkoon täytyi hakea tiedot tietokantaan tallennetuista kielistä ja sovellustyyppitaulukkoon tallennetuista sovellustyypeistä. Lisäksi haun tekemisen yhteydessä täytyi tietokannasta hakea kaikkiin hakuehtoihin linkitetyt työkalut.

Työkalun tietojen tallennus- ja päivityssivuille työkalujärjestelmä hakee tiedot sovellustyypeistä. Tallennussivusta ja hakusivusta poiketen päivityssivun sovellustyyppitaulukkoon tulee myös näkyviin päivitettävälle työkalulle ennestään merkatut sovellustyypit. Päivityssivulle haetaan sovellustyyppitietojen lisäksi myös työkalun ja työkaluun linkitettyjen tagien tiedot.

Historiasivulle tietoa noudetaan tietokannasta silloin, kun sinne on navigoitu jonkin työkalun päivityssivulta. Tällöin molempiin sivulle ilmestyviin alasvetovalikoihin haetaan tiedot historiatietojen tallentamiseen tarkoitetusta tool_aud-taulusta.

4 Tulokset

4.1 Työkalujärjestelmän prototyypin kehittämistyön tulokset

Sain toteutettua web-sovelluksena toimivan tietojärjestelmän prototyypin (työkalujärjestelmän), johon oli mahdollista sekä tallentaa että päivittää tietoa erilaisista työkaluista ja myös selata niitä. Kohtuullisen käytettävyyden saavuttamiseksi lisäsin työkalujärjestelmän web-lomakkeisiin validointi-toiminnon, jonka takia käyttäjä saa välitöntä palautetta kirjoittamiensa tietojen tallennuskelpoisuudesta.

Työkalujärjestelmään on mahdollista tallentaa toimeksiannossa mainitut työkalun nimi-, kuvaus- ja linkkitiedot. Myös työkalujen tietoihin on mahdollista liittää tieto mahdollisesta työkalun käyttämästä ohjelmointikielestä tagien muodossa. Lisäksi toteutin työkaluille päivitystoiminnon, jotta työkalujen tietoja voi tarvittaessa muokata esimerkiksi työkalun muuttuneen Internet-osoitteen takia.

Työkalujen hakutulosten näyttämiseen tehdyn sivun sain toteutettua Vaatimen avulla lähes täysin niin kuin olin toivonutkin. Samalla kyseinen toiminto näytti samalta kuin toimeksiannossa sillä poikkeuksella, että kielen valinta -toiminnon toteutin käyttäen alasvetovalikkoa valintaruutujen sijasta kohdassa 3.4.2 mainituista syistä.

Enversin odotettua helpomman käyttöönoton myötä toteutin yksinkertaisen työkalujen historian tallennus- ja katselutoiminnon, joka ei kuulunut työkalujärjestelmältä vaadittuihin ydintoiminnallisuuksiin, mutta sen toteuttaminen katsottiin hyödylliseksi.

Koska järjestelmä on prototyyppi, jäi siihen toteuttamatta joitakin virheentarkistus- ja lokiin kirjoitustoiminnallisuuksia, joita valmiista tietojärjestelmästä olisi hyvä löytyä. Toteuttamiani virheentarkistuksia työkalujärjestelmässä olivat erityisesti käyttäjäkokemukseen vaikuttavat, kuten väärin täytettyjen lomakkeiden, tarkistukset.

Työkalujärjestelmään jäi toteuttamatta mahdollisuus poistaa sen kautta tietueita tietokannasta. Käytännössä järjestelmään tallennettua tietoa voi poistaa vain tietokannanhallintajärjestelmän kautta jollakin muulla työkalulla kuin työkalujärjestelmällä. Syy kyseisen toiminnallisuuden pois jäämiseen oli aikapula. Uskoakseni mitään ongelmaa ei kyseisen ominaisuuden toteuttamisessa pitäisi tulla ainakaan työkalujärjestelmän kehittämiseen käytettyjen tekniikoiden osalta. Tämän vuoksi työkalujärjestelmän jatkokehittämisen ei pitäisi olla kohtuuttoman hankalaa.

4.2 Tulokset all-Java-paradigman ja siirrettävyyden kannalta

4.2.1 Tavoitteet

Työkalujärjestelmää toteuttaessa tavoitteekseni muodostui toteuttaa koko työkalujärjestelmä käyttäen pelkkää Java-ohjelmointikieltä. Tavoitteitani oli myös toteuttaa järjestelmä olio-suuntautuneesti ja tehdä siitä mahdollisimman riippumaton tietokannanhallintajärjestelmästä. Tavoitteiden saavuttamisen kannalta Vaadin- ja Hibernate-sovelluskehykset olivat keskeisessä roolissa. Vaadin ja Hibernate tekivät ns. likaisen työn ohjelmoijan puolesta kääntäen Java-koodin joissakin tilanteissa muiksi kieliksi, kuten JavaScript-, HTML- ja SQL-kieleksi.

4.2.2 Vaatimen edut

Vaadin mahdollisti sen, ettei työkalujärjestelmän käyttöliittymän kehittämisessä joutunut tekemään riviäkään koodia, johon olisi tarvinnut sekoittaa muita kieliä, kuten merkkauskieliä tai JavaScriptiä. Täten all-Java-lähestymistavan käyttäminen laajasti mahdollistui projektissa. Myös yleisimmät perustyyleihin kuuluvat CSS-tyyliasetukset pystyi toteuttamaan Java-koodin kautta käyttämällä perustyyleihin määriteltyjä vakioita. Koin, että Vaadin on hyvä ohjelmistokehys prototyyppitasoisen työkalujärjestelmän toiminnallisuuden toteuttamiseen.

4.2.3 Vaatimen haasteet ja ongelmat

Vaatimen haasteina ja ongelmina koin all-Java-järjestelmän kannalta sivujen tyyliin vaikuttamisen ilman itsetehtyjen tyylien tekemistä. Vaatimen version 6.8.6 sisään rakennetut teemat Runo, Reindeer ja Chameleon ovat mielestäni tyylikkäitä, mutta niiden avulla saa vain tietyn näköisiä sivuja tehtyä. Esimerkiksi Karelia-amk:n mahdollisen tyylikäsikirjan noudattaminen voi olla hankalaa edellä mainittuja tyylejä käyttäen. Chameleon-teemaan en tosin vielä ehtinyt tutustua kovinkaan tarkkaan sillä sitä ei ollut vielä versiossa 6.7.9, jota käytin suurimman osan projektista. Prototyypiltä ei vaadittu ulkonäöltä muuta kuin selkeyttä, joten käytännössä omalta osaltani ongelmallista oli lähinnä joidenkin yksittäisten komponenttien asettelu korkeussuunnassa samaan linjaan Java-ohjelmointikielellä.

Järjestelmän toteuttamisen aikana myös havaitsin, ettei kaikkea Javan ilmaisuvoimaa pystynyt hyödyntämään samalla tavoin kuin esimerkiksi työpöytäsovellusta tehdessä käyttäen Swingiä. Esimerkiksi käyttöliittymäkomponenttien käyttämisestä singleton-tekniikalla aiheutui ongelmia työkalujärjestelmän toiminnassa, jos sovelluksen aukaisi useampaan Internet-selaimeen.

Järjestelmän kehityksen aikana päädyin siihen lopputulokseen, että mikäli esimerkiksi mahdollinen jatkokehittäjä haluaa työkalujärjestelmästä ulkonäöllisesti miellyttävämmän ja suorituskykyisemmän, kannattanee siihen käyttää Vaadinta käyttäessä muitakin tekniikoita kuin pelkkää Javaa. Käyttöliittymän toteutukseen käytetyn sovelluskehyksen

ollessa Vaadin, on jatkokehittäjän melko helppoa parantaa sivujen ulkonäköä itse tehdyillä teemoilla käyttäen CSS-tekniikkaa (vrt. Grönroos 2012, 203–215).

4.2.4 Hibernaten edut

Aloittaessani toteuttamaan työkalujärjestelmää valitsin Hibernate-sovelluskehyksen, koska se ORM-sovelluskehyksenä (Object Relational Model Framework) mahdollisti olioiden persistoinnin tietokantaan hyvin vähällä määrällä itse kirjoitettua koodia. Näin ollen sain itse kirjoitetusta koodista huomattavasti oliomaisempaa kuin mitä olisin saanut ilman Hibernatea. Edellä mainittujen etujen lisäksi, saattoi Hibernaten toimintaan myös luottaa sen ollessa yksi käytetyimmistä ORM-sovelluskehyksistä ja täten hyvin testattu. Tämän vuoksi uskon, että Hibernatea käyttäessä säästyin myös monilta ohjelmointivirheiltä.

Hibernaten ja sen kautta käytettävän Criteria API:n avustamana työkalujärjestelmää toteuttaessa ei tarvinnut sekoittaa muita kieliä, kuten SQL- tai HQL-kieliä Java-koodin sekaan tietokantaoperaatioita tehdessä. Koodista sai niiden avulla myös hyvin oliopohjaista, mikä oli yksi tavoite projektia tehdessä.

Pystyin tiedon karttuessa välttymään käyttämästä XML-linkitystiedostoja, joita aluksi luulin välttämättömiksi Hibernatea soveltaessa. Havaitsin, että ne oli mahdollista korvata käyttämällä annotaatioita ja sain siten poistettua Hibernaten määrittelytiedostoa lukuun ottamatta kaiken itse merkatun XML-kielen projektista. Lisäksi sain koodista helpommin ylläpidettävää, koska muutoksia tarvitsi enää tehdä vain Java-tiedostoihin.

4.2.5 Hibernaten haasteet

Koko prototyyppiä ajatellen pelkän Javan käyttäminen Hibernaten kanssa vähensi melko paljon vaihtoehtoja ratkaista erilaiset ongelmat. Esimerkiksi olisin saattanut säästää aikaa, jos olisin käyttänyt HQL-kieltä Criteria API:n sijasta joihinkin hankaliin tietokantaoperaatioihin. Criteria API vaikuttaa yhtä hyvältä kuin HQL-kieli tietokantaoperaatioiden tekemiseen. Sen opetteleminen vaati kuitenkin enemmän aikaa kuin HQL:n opet-

teleminen. Suurin syy siihen oli se, että HQL:n käyttäminen poikkesi vähemmän SQL:n käyttämisestä kuin Criteria API:n opetteleminen. Toisaalta Criteria API:a käyttäen minun ei tarvinnut huolehtia valmistelluista lausekkeista itse, joten ainakin siinä säästin aikaa HQL-kielen käyttöön verrattuna.

Sekä HQL-kieltä että Criteria API:a käyttäessä huomasin, että projektissa käytetyn MySQL-tietokannan hallintajärjestelmän kohdalla piti jokaiselle luokalle tehdä asetus, joka annotaatioita käyttäen tehtiin kirjoittamalla @GeneratedValue(strategy = GenerationType.IDENTITY). Ymmärtääkseni kyseisen asetuksen saattaa joutua muuttamaan, jos vaihtaa tietokannanhallintajärjestelmää tai muuten työkalujärjestelmä ei välttämättä toimi oikein kaikissa tilanteissa. Tätä voi pitää pienenä kolauksena riippumattomuudelle tietokannanhallintajärjestelmästä, mutta toisaalta muutoksia ei tarvitse tehdä paljoa. HQL-kieli ja Criteria API eivät loppujen lopuksi tee tietokantaoperaatiosta standardi SQL-kieltä parempia tietokannan hallintajärjestelmäriippumattomuuden suhteen. Molemmissa vaikutti olevan mahdollista tehdä joitakin sellaisia operaatioita, jotka eivät toimi kaikissa tietokannanhallintajärjestelmissä. Pyrin kuitenkin välttämään sellaisten operaatioiden käyttöä työkalujärjestelmässä.

Pienenä all-Java-paradigman vastaisuutena koin sen, etten pystynyt määrittelemään Enversin kuuntelijoita (listeners) Java-koodin kautta. Siksi jouduin pelkästään niiden takia ottamaan takaisin käyttöön Hibernaten asetusten määrittelytiedoston, jossa kuuntelijoiden määrittelyyn käytetään XML-kieltä. Käsittääkseni Hibernaten versiosta 4 alkaen kyseistä ongelmaa ei enää pitäisi olla.

5 Pohdinta

Pohdittuani asiaa monelta kantilta näen työkalujärjestelmän kaltaisen prototyypin toteuttamisen kokonaan Javalla kohtalaisen hyvänä ideana. Ennen järjestelmän kehittämisen aloittamista olisi itselleni ollut huomattavasti helpompaa tehdä prototyypistä toivotunlainen käyttäen Javan lisäksi etenkin SQL-kieltä, koska Criteria API ei ollut minulle vielä tuttu. Valmiin järjestelmän toteuttamisessa pelkällä Javalla, etenkin Vaadin-sovelluskehyksen osalta, näen sen sijaan useita ongelmia. Vaadin-sivujen ulkonäköön

voi vaikuttaa merkittävästi CSS:n avulla, joten siitä pidättäytyminen voi vaikeuttaa sivujen saamista halutunlaiseen kuosiin. Vaatimen sisäänrakennetuista teemoista etenkin Reindeer oli ainakin työkalujärjestelmän prototyypin tekijän mieleen, joten ne mahdollisesti kelpaavat moneen muuhunkin järjestelmään sellaisenaan.

JavaScriptin käyttäminen avaisi uusia mahdollisuuksia muiden muassa Vaadin-liitännäisten kanssa, koska niiden joukosta löytyy esimerkiksi pelkästään selainpuolella toimiva CSValidation. CSValidation-liitännäisen avulla pitäisi olla mahdollista vähentää palvelinpuolen kuormaa.

Ohjelmistokehittäjänä tuntuisi ihanteelliselta tehdä esteettisesti miellyttävät ja kohtalaisen suorituskyvyn omaavat järjestelmät käyttämällä vain yhtä kieltä sen kaikkiin osa-alueisiin. Käyttämilläni työkaluilla ja pelkällä Java-ohjelmoinnilla se mielestäni onnistuu lähes toivotusti ja järjestelmät saa ulkonäöltään monin paikoin samalle tasolle kuin mitä se on esimerkiksi joissakin menestyvissä verkkokaupoissa tai pankkien verkkosivuilla. Vaatimen tapauksessa käyttäisin kuitenkin Javan lisäksi myös ainakin CSS:ää hioakseni sivujen ulkonäköä, koska se on minulle ennestään tuttu tekniikka. Pienellä vaivalla CSS:ällä voisi myös muuttaa mm. sivuston tekstien ulkonäköä huomattavasti. Vaatimen sisäänrakennetut teemat eivät välttämättä sovi useiden sivustojen tyyliin.

Dokumentoinnin kannalta näen all-Java-paradigman hyvänä ratkaisuna, sillä koko järjestelmän saa Javadoc API:in, jolloin kaikki tieto ohjelmistosta lähdekoodin osalta on samassa paikassa. Myös pelkkää Javaa tukevat UML-mallinnustyökalut, jotka tekevät UML-kaavioita Java-koodista, pystyvät all-Java-järjestelmässä saamaan käytännössä koko ohjelmiston yhteen kaavioon.

Tulevaisuudessa all-Java-filosofialla toteutettavat tietojärjestelmät voisivat olla varteen otettava vaihtoehto toteuttaa työkalujärjestelmän kaltaisia tietovarantoja. YouTuben kaltaisten RIA-sovellusten tekemiseen tarvinnee lisätä vähintäänkin Flash- tai HTML5-tekniikkaa Javan lisäksi. Toisaalta tulevaisuudessa Vaatimen kaltainen Java-sovelluskehys voi käyttää pinnan alla kaikkia tarvittavia HTML5-ominaisuuksia. Positiivisena puolena esimerkiksi ohjelmistoyrityksen kannalta voisi olla se, että Javaa osaavien ohjelmoijien löytäminen lienee Javan pitkään kestäneen suosion ansiosta koh-

talaisen helppoa ja työntekijältä ei tarvitse vaatia juurikaan tyypillisten web-sivuissa käytettävien kielten osaamista. Ohjelmoijilta toisaalta vaaditaan tiettyjen sovelluskehysten, kuten esimerkiksi Vaatimen ja Hibernaten osaamista. Lisäksi myös esimerkiksi SQL-kielelle löytyy sen tunnettuuden vuoksi huomattavasti enemmän dokumentteja ja muuta materiaalia Internetistä, mikä voi vähentää kiinnostusta all-Java-järjestelmiä kohtaan.

Lähteet

Free Pascal. 2012. FPC JVM. http://wiki.freepascal.org/FPC_JVM. 10.4.2013.

Google. 2008a. Developing with Google Web Toolkit. https://developers.google.com/web-toolkit/overview. 17.4.2013.

Google. 2008b. Google I/O 2008 - Dalvik Virtual Machine Internals. http://www.youtube.com/watch?v=ptjedOZEXPM. 17.4.2013.

Gosling, J., Joy B., Steele G., Bracha G. & Buckley, A. 2013. The Java® Language Specification, Java SE 7 Edition. http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf. 18.4.2013.

Grönroos, M. 2012. Book of Vaadin. https://vaadin.com/download/book-of-vaadin/vaadin-6/pdf/book-of-vaadin.pdf. 2.5.2013.

JBoss. 2004. Chapter 14. HQL: The Hibernate Query Language. http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html. 8.5.2013.

JBoss. 2005. Chapter 9. Criteria Queries. http://docs.jboss.org/hibernate/entitymanager/3.5/reference/en/html/querycriteria.html. 8.5.2013.

JBoss. 2010. Envers. http://www.jboss.org/envers. 8.5.2013.

JBoss. 2013. Hibernate Annotations. http://www.hibernate.org/about/annotations. 8.5.2013.

Lindholm, T., Yellin, F., Bracha, G. & Buckley, A. 2013. The Java Virtual Machine Specification. Oracle. http://docs.oracle.com/javase/specs/jvms/se7/jvms7.pdf. 25.3.2013.

Mozilla. 2011. HTTP Cachin FAQ. https://developer.mozilla.org/enUS/docs/HTTP_Caching_FAQ. 13.4.2013.

MySQL. 2013a. MySQL Community Edition. http://www.mysql.com/products/community/. 1.5.2013.

MySQL. 2013b. MySQL Query Browser. http://downloads.mysql.com/docs/query-browser-en.a4.pdf. 1.5.2013.

Oracle. 2011a. Abstract Window Toolkit (AWT). http://docs.oracle.com/javase/6/docs/technotes/guides/awt/. 17.4.2013.

Oracle. 2011b. Swing (Java™ Foundation Classes). http://docs.oracle.com/javase/6/docs/technotes/guides/swing/index.html. 17.4.2013.

Oracle. 2011c. Class ArrayList<E>. http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html. 25.4.2013.

Oracle. 2012a. Java (TM) Platform, Standard Edition Runtime Environment Version 7. http://www.oracle.com/technetwork/java/javase/jre-7-readme-430162.html. 25.3.2013.

Oracle. 2012b. Java Platform Standard Edition 7 Documentation. http://docs.oracle.com/javase/7/docs/index.html. 26.3.2013.

Oracle. 2012c. JAR File Overview. http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jarGuide.html. 27.3.2013.

Oracle. 2012d. The Java Programming Language Platforms. http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. 19.4.2013.

Oracle. 2013a. About the Java Technology.
http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html.
28.4.2013.
Oracle. 2013b. The Java™ Programming Language Compiler, javac.
http://docs.oracle.com/javase/7/docs/technotes/guides/javac/index.html.
17.4.2013.
Oracle. 2013c. Java ME Technology Overview.
http://www.oracle.com/technetwork/java/javame/java-me-overview-
402920.html. 25.4.2013.
Oracle 2013d. The Java EE 6 Tutorial.
http://docs.oracle.com/javaee/6/tutorial/doc/geysj.html. 8.5.2013.
Oracle. 2013e. Javadoc Tool.
http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-
135444.html. 17.4.2013.
SimpleCaptcha. 2013. SimpleCaptcha. http://simplecaptcha.sourceforge.net/index.html.
2.5.2013.
The Apache Software Foundation. 2013. Apache Tomcat. http://tomcat.apache.org/.
1.5.2013.
Tsagklis, I. 2010. Adding CAPTCHA to your GWT application.
http://www.javacodegeeks.com/2010/06/add-captcha-gwt-application.html.
23.5.2013.
Vaadin. 2012a. Release Notes for Vaadin Framework 6.8.6.
http://vaadin.com/download/release/6.8/6.8.6/release-notes.html. 2.5.2013.
Vaadin. 2012b. Welcome to Vaadin Directory. https://vaadin.com/directory/help.
8.5.2013.
Washington, P. 2010. Java Hibernate Tutorial Part 5 - Create table from Class.
http://www.youtube.com/watch?v=ReAZmA83Myg. 6.5.2013.